T2: Multiagent Reinforcement Learning (MARL)

May 7, 2013 - AAMAS'13

Presenters

- Daan Bloembergen
- Tim Brys
- Daniel Hennes
- Michael Kaisers
- Mike Mihaylov
- Karl Tuyls

Schedule

- Fundamentals of multi-agent reinforcement learning
  - **09:00 - 10:30**, Daan Bloembergen and Daniel Hennes
- Research applications of MARL
  - **11:00 - 12:30, 13:30 - 14:00**, Tim Brys and Mihail Mihaylov
- Dynamics of learning in strategic interactions
  - **14:00 - 14:45**, Michael Kaisers
- A framework for multi-agent systems
  - **14:45 - 15:00**, Michael Kaisers
- Practical demos, discussion and questions
  - **15:00 - 15:30, 16:00 - 17:00**
- Optional: joint ALA & MSDM panel
  - **17:00 - 18:00**

Who are you?

We would like to get to know our audience!

Fundamentals of Multi-Agent Reinforcement Learning

Daan Bloembergen and Daniel Hennes

Outline (1)

Single Agent Reinforcement Learning

- Markov Decision Processes
  - Value Iteration
  - Policy Iteration
- Algorithms
  - Q-Learning
  - Learning Automata

Outline (2)

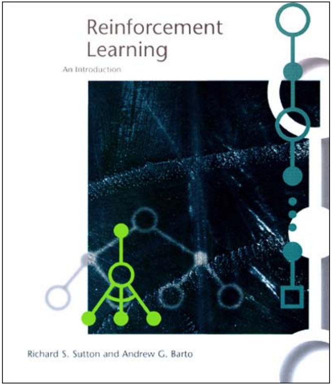
Multiagent Reinforcement Learning

- Game Theory
- Markov Games
  - Value Iteration
- Algorithms
  - Minimax-Q Learning
  - Nash-Q Learning
  - Other Equilibrium Learning Algorithms
  - Policy Hill-Climbing

Part I:
Single Agent Reinforcement Learning

Richard S. Sutton and Andrew G. Barto
Reinforcement Learning: An Introduction
MIT Press, 1998

Available on-line for free!



Why reinforcement learning?

Based on ideas from psychology

- Edward Thorndike's **law of effect**
  - Satisfaction strengthens behavior, discomfort weakens it
- B.F. Skinner's **principle of reinforcement**
  - Skinner Box: train animals by providing (positive) feedback

Learning by interacting with the environment

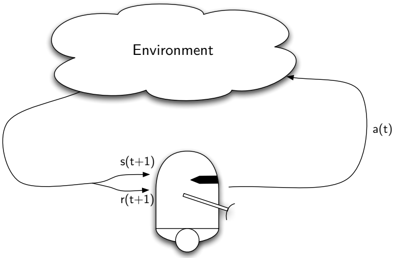
Why reinforcement learning?

Control theory

- Design a controller to minimize some measure of a dynamical systems's behavior
- Richard Bellman
  - Use system state and value functions (optimal return)
  - **Bellman equation**
- Dynamic programming
  - Solve optimal control problems by solving the Bellman equation

These two threads came together in the 1980s, producing the modern field of reinforcement learning

The RL setting



- Learning from interactions
- Learning what to do - **how to map situations to actions** - so as to maximize a numerical reward signal

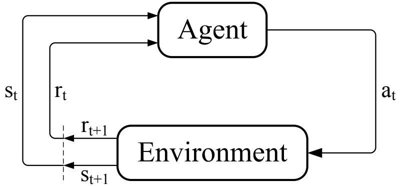
Key features of RL

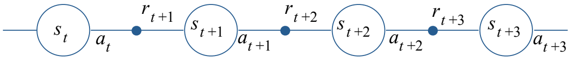
- Learner is **not** told which action to take
- Trial-and-error approach
- Possibility of **delayed reward**
  - Sacrifice short-term gains for greater long-term gains
- Need to balance **exploration** and **exploitation**
- In between **supervised** and **unsupervised** learning

The agent-environment interface

Agent interacts at discrete time steps $t = 0, 1, 2, \ldots$

- Observes state $s_t \in S$
- Selects action $a_t \in A(s_t)$
- Obtains immediate reward $r_{t+1} \in \Re$
- Observes resulting state s_{t+1}





Elements of RL

- Time steps need not refer to fixed intervals of real time
- **Actions** can be
  - low level (voltage to motors)
  - high level (go left, go right)
  - "mental" (shift focus of attention)
- **States** can be
  - low level "sensations" (temperature, (x, y) coordinates)
  - high level abstractions, symbolic
  - subjective, internal ("surprised", "lost")
- The **environment** is not necessarily known to the agent

Elements of RL

- **State transitions** are
  - changes to the internal state of the agent
  - changes in the environment as a result of the agent's action
  - can be nondeterministic
- **Rewards** are
  - goals, subgoals
  - duration
  - ...

Learning how to behave

- The agent's **policy** π at time t is
  - a mapping from states to action probabilities
  - $\pi_t(s, a) = P(a_t = a | s_t = s)$
- Reinforcement learning methods specify **how** the agent changes its policy as a result of experience
- Roughly, the agent's goal is to get **as much reward** as it can over the long run

The objective

Suppose the sequence of rewards after time t is

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

- The goal is to maximize the **expected return** $E\{R_t\}$ for each time step t
- **Episodic tasks** naturally break into episodes, e.g., plays of a game, trips through a maze

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

The objective

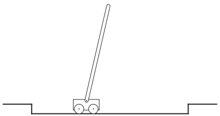
- **Continuing tasks** do not naturally break up into episodes
- Use **discounted return** instead of total reward

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where γ, $0 \leq \gamma \leq 1$ is the **discount factor** such that

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

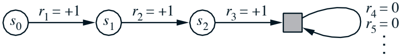
Example: pole balancing



- As an **episodic** task where each episode ends upon failure
  - reward = +1 for each step before failure
  - return = number of steps before failure
- As a **continuing** task with discounted return
  - reward = -1 upon failure
  - return = $-\gamma^k$, for k steps before failure
- In both cases, return is maximized by avoiding failure for as long as possible

A unified notation

- Think of each episode as ending in an **absorbing state** that always produces a reward of zero



- Now we can cover both episodic and continuing tasks by writing

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Markov decision processes

- It is often useful to a assume that all relevant information is present in the current state: **Markov property**

$$P(s_{t+1}, r_{t+1} | s_t, a_t) = P(s_{t+1}, r_{t+1} | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0)$$

- If a reinforcement learning task has the Markov property, it is basically a **Markov Decision Process (MDP)**
- Assuming finite state and action spaces, it is a finite MDP

Markov decision processes

An MDP is defined by

- **State and action sets**
- One-step dynamics defined by state **transition probabilities**

$$\mathcal{P}^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- **Reward probabilties**

$$\mathcal{R}^a_{ss'} = E(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$$

Value functions

- When following a fixed policy π we can define the **value** of a state s under that policy as

$$V^{\pi}(s) = E_{\pi}(R_t|s_t = s) = E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s)$$

- Similarly we can define the value of taking action a in state s as

$$Q^{\pi}(s, a) = E_{\pi}(R_t|s_t = s, a_t = a)$$

Value functions

- The value function has a particular recursive relationship, defined by the **Bellman equation**

$$V^{\pi}(s) = \sum_{a} \pi(s, a) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')]$$

- The equation expresses the recursive relation between the value of a state and its successor states, and averages over all possibilities, weighting each by its probability of occurring

Optimal policy for an MDP

- We want to find the policy that maximizes long term reward, which equates to finding the optimal value function

$$V^*(s) = \max_\pi V^\pi(s) \qquad \forall s \in S$$
$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \qquad \forall s \in S, a \in A(s)$$

- Expressed recursively, this is the **Bellman optimality equation**

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi *}(s,a) \\ &= \max_{a \in A(s)} \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')] \end{aligned}$$

Solving the Bellman equation

- We can find the **optimal policy** by solving the Bellman equation
  - Dynamic Programming
- Two approaches:
  - Iteratively improve the value function: **value iteration**
  - Iteratively evaluate and improve the policy: **policy iteration**
- Both approaches are proven to converge to the optimal value function

Value iteration

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Policy iteration

- Often the optimal policy has been reached long before the value function has converged
- Policy iteration calculates a new policy **based on the current value function**, and then calculates a new value function based on this policy
- This process often converges faster to the optimal policy

Policy iteration

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad b \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   $\quad$ If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Learning an optimal policy online

- Both previous approaches require to know the dynamics of the environment
- Often this information is not available
- Using **temporal difference (TD)** methods is one way of overcoming this problem
  - Learn directly from raw experience
  - No model of the environment required (model-free)
  - E.g.: **Q-learning**
- Update predicted state values based on new observations of immediate rewards and successor states

Q-learning

- Q-learning updates state-action values based on the immediate reward and the optimal expected return

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

- Directly learns the optimal value function independent of the policy being followed
    - In contrast to on-policy learners, e.g. **SARSA**
- Proven to converge to the optimal policy given "sufficient" updates for each state-action pair, and decreasing learning rate α [Watkins92]

Q-learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$
        $s \leftarrow s'$;
    until s is terminal

Action selection

- How to select an action based on the values of the states or state-action pairs?
- Success of RL depends on a **trade-off**
  - Exploration
  - Exploitation
- **Exploration** is needed to prevent getting stuck in local optima
- To ensure convergence you need to **exploit**

Action selection

Two common choices

- **ϵ-greedy**
  - Choose the best action with probability $1 - \epsilon$
  - Choose a random action with probability ϵ
- **Boltzmann exploration** (softmax) uses a temperature parameter τ to balance exploration and exploitation

$$\pi_t(s, a) = \frac{e^{Q_t(s,a)/\tau}}{\sum_{a' \in A} e^{Q_t(s,a')/\tau}}$$

pure exploitation $\quad 0 \leftarrow \tau \rightarrow \infty \quad$ pure exploration

Learning automata

- **Learning automata** [Narendra74] directly modify their policy based on the observed reward (policy iteration)
- Finite action-set learning automata learn a policy over a finite set of actions

$$\pi'(a) = \pi(a) + \begin{cases} \alpha r(1-\pi(a)) - \beta(1-r)\pi(a) & \text{if } a = a_t \\ -\alpha r \pi(a) + \beta(1-r)[(k-1)^{-1} - \pi(a)] & \text{if } a \neq a_t \end{cases}$$

  where $k = |A|$, and α and β are reward and penalty parameters respectively, and $r \in [0,1]$
- **Cross learning** is a special case where $\alpha = 1$ and $\beta = 0$

Networks of learning automata

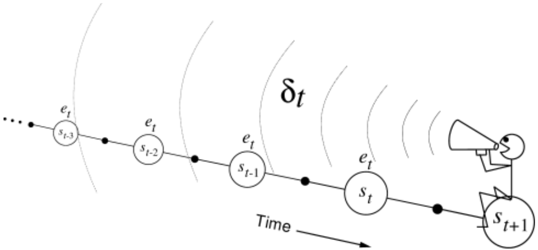
- A single learning automaton ignores any state information
- In a **network of learning automata** [Wheeler86] control is passed on from one automaton to another
  - One automaton $\mathcal{A}$ is active for each state
  - The immediate reward r is replaced by the average cumulative reward $\bar{r}$ since the last visit to that state

$$\bar{r}_t(s) = \frac{\Delta r}{\Delta t} = \frac{\sum_{i=l(s)}^{t-1} r_i}{t - l(s)}$$

where $l(s)$ indicates in which time step state s was last visited

Extensions

- Multi-step TD: **eligibility traces**
  - Instead of observing one immediate reward, use n consecutive rewards for the value update
  - Intuition: your current choice of action may have implications for the future
  - State-action pairs are eligible for future rewards, with more recent states getting more credit



Extensions

- **Reward shaping**
  - Incorporate domain knowledge to provide additional rewards during an episode
  - Guide the agent to learn faster
  - (Optimal) policies preserved given a potential-based shaping function [Ng99]
- **Function approximation**
  - So far we have used a tabular notation for value functions
  - For large state and actions spaces this approach becomes intractable
  - Function approximators can be used to generalize over large or even continuous state and action spaces

Questions so far?

Part II:
Multiagent Reinforcement Learning

Preliminaries: Fundamentals of Game Theory

Game theory

- Models **strategic interactions** as games
- In **normal form games**, all players simultaneously select an action, and their joint action determines their individual payoff
  - One-shot interaction
  - Can be represented as an n-dimensional payoff matrix, for n players
- A player's **strategy** is defined as a probability distribution over his possible actions

Example: Prisoner's Dilemma

- Two prisoners (A and B) commit a crime together
- They are questioned separately and can choose to confess or deny
  - If both confess, both prisoners will serve 3 years in jail
  - If both deny, both serve only 1 year for minor charges
  - If only one confesses, he goes free, while the other serves 5 years

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

Example: Prisoner's Dilemma

- What should they do?
- If both deny, their total penalty is lowest
  - But is this individually rational?
- Purely selfish: regardless of what the other player does, confess is the optimal choice
  - If the other confesses, 3 instead of 5 years
  - If the other denies, free instead of 1 year

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

Solution concepts

- **Nash equilibrium**
  - Individually rational
  - No player can improve by unilaterally changing his strategy
  - Mutual confession is the only Nash equilibrium of this game
- Jointly the players could do better
  - **Pareto optimum**: there is no other solution for which all players do at least as well and at least one player is strictly better off
  - Mutual denial Pareto dominates the Nash equilibrium in this game

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

Types of games

- **Competitive** or zero-sum
  - Players have opposing preferences
  - E.g. Matching Pennies
- **Symmetric games**
  - Players are identical
  - E.g. Prioner's Dilemma
- **Asymmetric games**
  - Players are unique
  - E.g. Battle of the Sexes

Matching Pennies

	H	T
H	+1, -1	-1, +1
T	-1, +1	+1, -1

Prisoner's Dilemma

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

Battle of the Sexes

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

Part II: Multiagent Reinforcement Learning

MARL: Motivation

- MAS offer a solution paradigm that can cope with complex problems
- Technological challenges require decentralised solutions
  - Multiple autonomous vehicles for exploration, surveillance or rescue missions
  - Distributed sensing
  - Traffic control (data, urban or air traffic)
- Key advantages: Fault tolerance and load balancing
- But: highly dynamic and nondeterministic environments!
- Need for adaptation on an individual level
- Learning is crucial!

MARL: From single to multiagent learning

- Inherently more challenging
- Agents interact with the environment and each other
- Learning is simultaneous
- Changes in strategy of one agent might affect strategy of other agents
- Questions:
  - One vs. many learning agents?
  - Convergence?
  - Objective: maximise common reward or individual reward?
  - Credit assignment?

Independent reinforcement learners

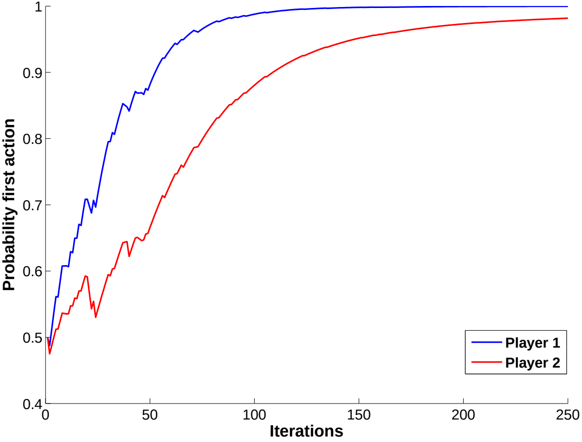
- Naive extension to multi agent setting
- Independent learners mutually ignore each other
- Implicitly perceive interaction with other agents as noise in a stochastic environment

Learning in matrix games

- Two Q-learners interact in Battle of the Sexes
  - $\alpha = 0.01$
  - Boltzmann exploration with $\tau = 0.2$
- They only observe their immediate reward
- Policy is gradually improved

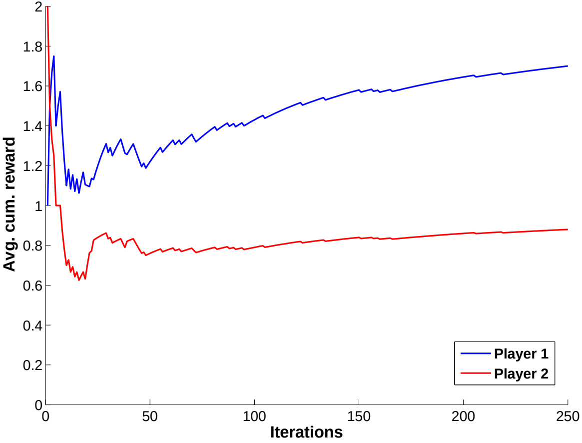
	B	S
B	2, 1	0, 0
S	0, 0	1, 2

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

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/payoff for player i
- $\mathcal{P}$: transition function

The payoff function $\mathcal{R}^i : S \times A^1 \times \cdots \times A^n \mapsto \mathbb{R}$ maps the joint action $a = \langle a^1 \ldots a^n \rangle$ to an immediate payoff value for player i.

The transition function $\mathcal{P} : S \times A^1 \times \cdots \times A^n \mapsto \triangle(S)$ determines the probabilistic state change to the next state s_{t+1}.

Value iteration in Markov games

Single agent MDP:

$$V^*(s) = \max_{a \in A(s)} Q^{\pi*}(s, a)$$
$$= \max_{a \in A(s)} \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

2-player zero-sum stochastic game:

$$Q^*(s, \langle a^1, a^2 \rangle) = \mathcal{R}(s, \langle a^1, a^2 \rangle) + \gamma \sum_{s' \in S} \mathcal{P}_{s'}(s, \langle a^1, a^2 \rangle) V^*(s')$$

$$V^*(s) = \max_{\pi \in \triangle(A^1)} \min_{a^2 \in A^2} \sum_{a^1 \in A^1} \pi_{a^1} Q^*(s, \langle a^1, a^2 \rangle)$$

Minimax-Q

- Value iteration requires knowledge of the reward and transition functions
- Minimax-Q [Littman94]: learning algorithm for zero-sum games
- Payoffs balance out, each agent only needs to observe its own payoff
- Q is a function of the joint action:

$$Q(s, \langle a^1, a^2 \rangle) = \mathcal{R}(s, \langle a^1, a^2 \rangle) + \gamma \sum_{s' \in S} \mathcal{P}_{s'}(s, \langle a^1, a^2 \rangle) V(s')$$

- A joint action learner (JAL) is an agent that learns Q-values for joint actions as opposed to individual actions.

Minimax-Q (2)

Update rule for agent 1 with reward function $\mathcal{R}_t$ at stage t:

$$Q_{t+1}(s_t, \langle a_t^1, a_t^2 \rangle) = (1 - \alpha_t)\; Q_t(s_t, \langle a_t^1, a_t^2 \rangle) + \alpha_t \left[\mathcal{R}_t + \gamma\, V_t(s_{t+1})\right]$$

The value of the next state $V(s_{t+1})$:

$$V_{t+1}(s) = \max_{\pi \in \triangle(A^1)} \; \min_{a^2 \in A^2} \sum_{a^1 \in A^1} \pi_{a^1}\, Q_t(s, \langle a^1, a^2 \rangle)\,.$$

Minimax-Q converges to Nash equilibria under the same assumptions as regular Q-learning [Littman94]

Nash-Q learning

- Nash-Q learning [Hu03]: joint action learner for general-sum stochastic games
- Each individual agent has to estimate Q values for all other agents as well
- Optimal Nash-Q values: sum of immediate reward and discounted future rewards under the condition that all agents play a specified Nash equilibrium from the next stage onward

Nash-Q learning (2)

Update rule for agent i:

$$Q^i_{t+1}(s_t, \langle a^1, \dots, a^n \rangle) = (1 - \alpha_t)\, Q(s_t, \langle a^1, \dots, a^n \rangle) \\ + \alpha_t \left[\mathcal{R}_t + \gamma\, Nash V^i_t(s_{t+1}) \right]$$

A Nash equilibrium is computed for each stage game $\left(Q^1_t(s_{t+1}, \cdot), \dots, Q^n_t(s_{t+1}, \cdot)\right)$ and results in the equilibrium payoff $Nash V^i_t(s_{t+1}, \cdot)$ to agent i

Agent i uses the same update rule to estimate Q values for all other agents, i.e., $Q^j\ \forall j \in \{1, \dots, n\} \backslash i$

Other equilibrium learning algorithms

- Friend-or-Foe Q-learning [Littman01]
- Correlated-Q learning (CE-Q) [Greenwald03]
- Nash bargaining solution Q-learning (NBS-Q) [Qiao06]]
- Optimal adaptive learning (OAL) [Wang02]
- Asymmetric-Q learning [Kononen03]

Limitations of MARL

- Convergence guarantees are mostly restricted to stateless repeated games
- ... or are inapplicable in general-sum games
- Many convergence proofs have strong assumptions with respect to a-priori knowledge and/or observability
- Equilibrium learners focus on stage-wise solutions (only indirect state coupling)

Summary

In a multi-agent system

- be aware what information is available to the agent
- if you can afford to try, just run an algorithm that matches the assumptions
- proofs of convergence are available for small games
- new research can focus either on engineering solutions, or advancing the state-of-the-art theories

Questions so far?

Thank you!

Daan Bloembergen | daan.bloembergen@gmail.com
Daniel Hennes | daniel.hennes@gmail.com