Multi-agent reinforcement learning for new generation control systems

Manuel Graña[1,2]; Borja Fernandez-Gauna[2]

[1]ENGINE centre, Wroclaw Technological University; [2]Computational Intelligence Group (www.ehu.eus/ccwintco)
University of the Basque Country (UPV/EHU)

IDEAL, 2015

Overall view of the talk

- Comment on Reinforcement Learning and Multi-Agent Reinforcement Learning
- Not a tutorial
- Our own contributions in the last times (mostly Borja's)
    - improvements on RL avoiding traps
    - a "new" coordination mechanism in MARL : D-RR-QL
- A glimpse on a promising avenue of research in MARL

Contents

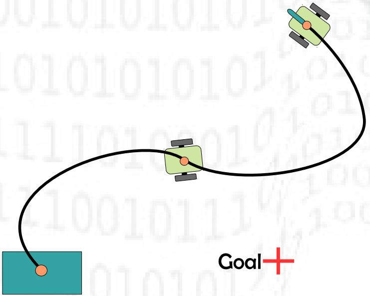
Contents

Motivation

- Goals of innovation in control systems:
  - attain an *acceptable* control system
    - when system's dynamics are not fully understood or precisely modeled
    - when training feedback is sparse or minimal
  - autonomous learning
    - adaptability to changing environments
  - distributed controllers robust to component failures
    - large multicomponent systems
  - Minimal human designer input

Example

- Multi-robot transportation of a hose
  - non-linear dyamical strong interactions trough an elastic deformable link
  - hard constraints:
    - robots could drive over the hose, overstretch it, collide, ...
    - sources of uncertainty: hose position, hose weight and intrinsic forces (elasticity)



Reinforcement Learning for controller design

- Reinforcement Learning
    - agent-environment interaction
    - learning action policies from rewards
        - time delayed rewards
        - almost unsupervised learning
- Advantages:
    - Designer does not specify (input, output) training samples
        - rewards are positive upon reaching the task completion
    - Model free
    - Autonomous adaptation to slowly changing conditions
        - exploitation vs. exploration dilemma

Contents

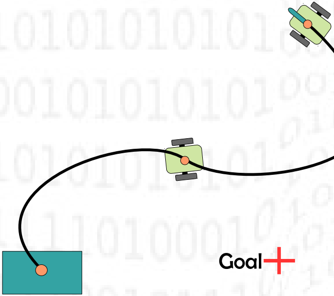
Contents

Markov Decision Process (MDP)

- Single-agent environment interaction modeled as Markov Decision Processes $\langle S, A, P, R \rangle$
  - S: the set of states the system can have
  - A: the set of actions from which the agent can choose
  - P: the transition function
  - R: the reward function

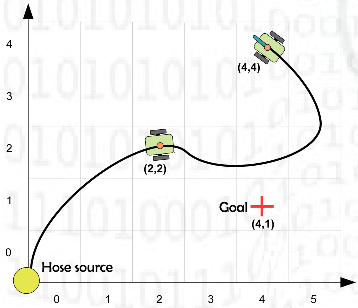
Single-agent approach

- The simplest approach to the multirobot hose transportation:
  - a unique central agent learning how to control all robots



The set of states: S

- Simple state model
  - S is a set of discrete states
  - State: discretized spatial position of the two robots. e.g.: $\langle (2,2),(4,4) \rangle$.
    - In a 5×4 grid, total amount of 20^2 states



Single-Agent MDP

Observation

Single-Agent MDP can deal with multicomponent systems

- State space is the product space of component state spaces
- Action space is the space of joint actions
- Dynamics of all components are pull together
- Reward is system global
- Equivalent to a centralized monolithic controller

The set of actions: A

- Discrete set of actions for each robot:
  - $A_1 = \{up_1, down_1, left_1, right_1\}$
  - $A_2 = \{up_2, down_2, left_2, right_2\}$
- If we want the agent to move both robots at the same time, the set of joint-actions is $A = A_1 \times A_2$:
  - $A = \{up_1/up_2, up_1/down_2, \ldots, down_1/up_2, down_1/down_2, ...\}$
- 16 different joint-actions

The transition function: P

- Defines the state transitions induced by action execution
  - Deterministic (state-action mapping): $P : S, A \rightarrow S$;
    - $s' = P(s,a)$ s' observed after a is executed in s.
  - Stochastic (probability distribution): $P : S, A, S \rightarrow [0,1]$
    - $p(s'|s,a)$ probability of observing s' after a is executed in s.

The reward function: R

- This function returns the environment's evaluation of either
  - the last agent's decision: i.e. action executed $R : S \times A \rightarrow \mathbb{R}$
  - state reached: $R : S \rightarrow \mathbb{R}$
- It is the objective function to be maximized
  - given by the system designer
- A reward function for our hose transportation task:

$$R(s) \begin{cases} 1 & \textit{if } s = \textit{Goal} \\ 0 & \textit{otherwise} \end{cases}$$

Learning

- The goal of the agent is to learn a policy $\pi(s)$ that maximizes the accumulated expected rewards
- Each time-step:
  - The agent observes the state s
  - Applying policy π, it chooses and executes action a
  - A new state s' is observed and reward r is received by the agent
  - The agent "learns" by updating the estimation of the value of states and actions

Q-Learning

- State value function : expected rewards from state s following policy $\pi(s)$:

$$V^{\pi}(s) = E^{\pi}\left\{\sum_{t=0}^{\infty} \gamma^t r_t \mid s = s_t\right\}$$

  - discount parameter γ
    - weight higher immediate rewards than future ones

- state-action value function $Q(s,a)$:

$$Q^{\pi}(s,a) = E^{\pi}\left\{\sum_{t=0}^{\infty} \gamma^t r_t \,|\, s = s_t \wedge a = a_t\right\}$$

Q-Learning

- Q-Learning : iterative estimation of Q-values :

$$Q_t(s,a) = (1-\alpha)\, Q_{t-1}(s,a) + \alpha \cdot \left[r_t + \gamma \cdot \max_{a'} Q_{t-1}(s', a') \right],$$

where α is the learning gain.

- Tabular representation : store value of each state-action pair $(|S| \cdot |A|)$
  - In our example, with 2 robots (20 states) and 4 actions per robot, the Q-table size : $20 \cdot 4^2$

Action-selection policy

- Convergence: Q-learning converges to the optimal Q-table
  - iff all possible state-action pairs are visited infinitely often
- Exploration: requires trying suboptimal actions to gather information (convergence)
  - $\varepsilon - greedy$ action selection policy:

$$\pi_{\varepsilon}(s) = \begin{cases} random\,action & with\,probability\,\varepsilon \\ \arg\max_{a \in A} Q(s,a) & with\,probability\,1-\varepsilon \end{cases}$$

- Exploitation: selects action $a^* = \max_a Q(s,a)$

Learning

Observation

- Learning often requires the repetition of experiments
- Repetitions often imply simulation is the only practical way
- Autonomous learning implies exploration
  - non-stationarity asks for permanent exploration

Physical constraints

- Robotic control tasks ofter present physical constraints : undesirable termination state-actions (UTS)
  - experiment (simulation) terminated without learning anything positive
- Linked MCRS physical constraints:
  - Overstrechting the hose: elastic until breaking point
  - Driving over the hose
  - Colliding with each other
  - Get outside the working space

Reward function

- Teach the agent to avoid breaking physical constraints =>
  - introduce those constraints in the reward function
    - negative rewards

$$R(s)\begin{cases} 1 & \textit{if } s = Goal \\ -1 & \textit{if physical constraint broken} \\ 0 & \textit{otherwise} \end{cases}$$

Reducing Learning complexity

- Learning time conditioned by
  - theoretical convergence conditions
  - time to perform/simulate each action/experiment
  - failed experiments in overconstrained systems
- Space requirements
  - state-action explosion in multicomponent systems

Our work: L-MCRS

- We use Geometrically Exact Dynamic Splines (GEDS) to simulate the hose dynamics
  - The simulation time for a single step with only two robots is about 45 seconds
- When a physical constraint is broken, the system must be reset

Our work: L-MCRS

- We have presented several techniques to make learning L-MCRS control more efficient:
  - Modular Action-State Vetoes
  - Undesired State-Action Prediction
  - Transfer Learning using Partially Constrained Models
  - Functional approximations: Actor-Critic
  - Distributed Round-Robin Q-Learning –> Multiagent Reinforcement Learning

Contents

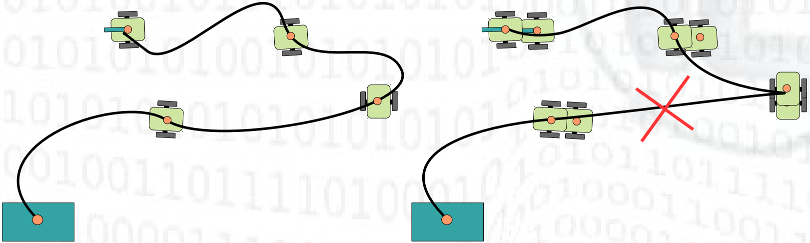
Modular State-Action Vetoes

- Undesired Terminal States (UTS) are vetoed[1]
- Rationale:
  - UTS do not need to be revisited
  - Not all state variables drive to the UTS
    - Decomposable detection of UTS $- >$ modularity
- Achieving learning speed-up
  - Increased space exploration

[1]B. Fernandez-Gauna; JM Lopez-Guede; I Etxeberria-Agiriano; I Ansoategi; M Graña Reinforcement Learning endowed with safe veto policies to learn the control of L-MCRS Information Sciences Volume 317, 1 October 2015, Pages 25–47 [8] DOI 10.1016/j.ins.2015.04.005

Modular State-Action Vetoes

- Example:
  - If the system executes action $\{left_1, left_2, up_3, left_4\}$, the hose is overstretched and possibly broken



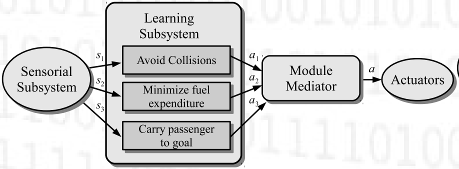
Modular State-Action Vetoes

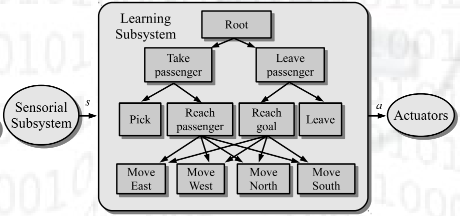
- Question: would it have been overstretched if the first two robots had another position?
  - Physical constraints are related with a subset of the state variables
  - The agent can then veto state-actions on the basis of information only from this subset of state variables

Modular State-Action Vetoes

Observation

Single-Agent internal logic may be modular





Modular State-Action Vetoes

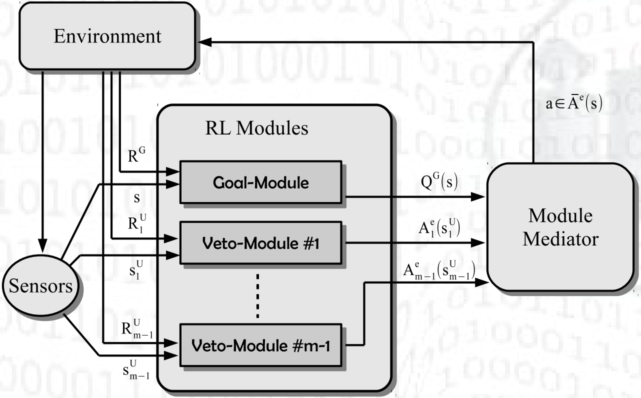
- We decompose the reward signal into

$$R(s) = R^{G}(s) + \sum_{i=1}^{m} R_i^{U}(s),$$

  - positive reward $R^{G}(s)$ and
  - m negative rewards R_i^U, each of them triggered when a certain class of physical constraint is broken
- We determine automatically the relevance of each state variable for each R_i^U
- Reward function partitions S into three disjoint subspaces: goal states G, transition states T, and UTS U,

$$\begin{aligned} G &= \{s \mid s \in S, R(s) > 0\}, \\ T &= \{s \mid s \in S, R(s) = 0\}, \\ U &= \{s \mid s \in U, R(s) < 0\}. \end{aligned}$$

Modular State-Action Vetoes



Modular State-Action Vetoes

- Each time R_i^U is triggered, the last action executed is vetoed on the *i*-th module's state subspace (several states at the same time)
- Safe action repertoire A_i^e is defined in its own state subspace as:

$$A_i^e\left(s_i^U\right)=\left\{a \,\middle|\, a\in A\wedge\left(\sum_{s'\in[U]_{S_i^U}} P_i\left(s_i^U,a,s'\right)>0\right)\right\},$$

- State safe action repertoire estimated as

$$\bar{A}^e(s)=\bigcap_{i=1\ldots m-1}\bar{A}_i^e\left([s]_{S_i^U}\right).$$

Modular State-Action Vetoes

- Safe vetoed exploration policies

$$\hat{\pi}_{\varepsilon-greedy}(s,a,\varepsilon)=\begin{cases}0 & Veto(s,a)\\ \frac{\varepsilon}{|\bar{A}^e(s)|} & \neg Veto(s,a)\wedge a\neq \underset{a'\notin A^e(s)}{\arg\max}\left\{Q^G\left([s]_{S^G},a'\right)\right\}\\ 1-\varepsilon & \neg Veto(s,a)\wedge a= \underset{a'\notin A^e(s)}{\arg\max}\left\{Q^G\left([s]_{S^G},a'\right)\right\}\end{cases}, \tag{1}$$

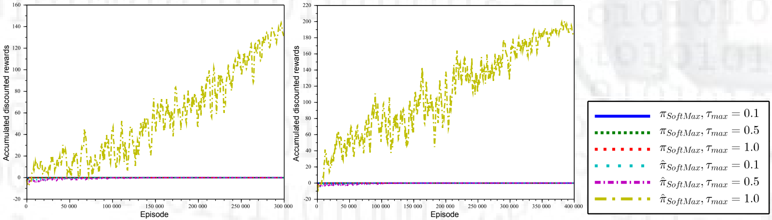
Modular State-Action Vetoes

Theorem

Let $< S, A, P, R >$ be a Monolithic MDP decomposed and trained as a Safe-MSAV Modular MDP $\left[\langle S, A, P, R^G\rangle, \{\langle S_i^U, A, P, R_i^U\rangle\}_{i=1}^{m-1}\right]$. Under the stochastic gradient convergence conditions and assuming infinite visits along infinite exploration time to all state-action pairs in $T \times A$, Q-Learning with Veto-based action selection algorithms will converge to the optimal Q-values for the restricted state space MDP $\langle T \cup G, A^e(s), P, R\rangle$.

Modular State-Action Vetoes

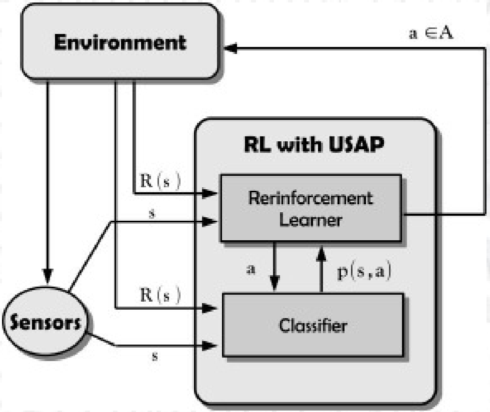
- faster learning : focus on learning the Q-value of safe state-actions
- Some results from : single-agent Q-Learning with/without MSAV



Contents

Undesired State-Action Prediction (USAP)

- Unsafe actions by Supervised Prediction (USAP) by Machine Learning[2]



[2] Borja Fernandez-Gauna; Ion Marques; Manuel Graña Undesired State-Action Prediction in Multi-Agent Reinforcement Learning. Application to Multicomponent Robotic System control Information Sciences (2013) 232:309–324

Undesired State-Action Prediction (USAP)

- The USAP module training samples are of the form $\langle s, a, c \rangle$, where $c \in \{SAFE, UNSAFE\}$
- After training, the USAP predicts the probability of unsafeness

$$p(s,a) = \sum_{s' \in U} P(s,a,s')$$

$$A^s(s) = \{a \in A \mid p(s,a) < 0.5\}$$

$$\pi_{\varepsilon}^{USAP}(s,a) = \begin{cases} 0 & \text{if } a \notin A^s(s) \\ \frac{\varepsilon}{|A^s(s)|} & a \neq \arg \max\limits_{a' \in A^s(s)} \{Q(s,a')\} \\ 1-\varepsilon & \text{otherwise} \end{cases},$$

.

Undesired State-Action Prediction

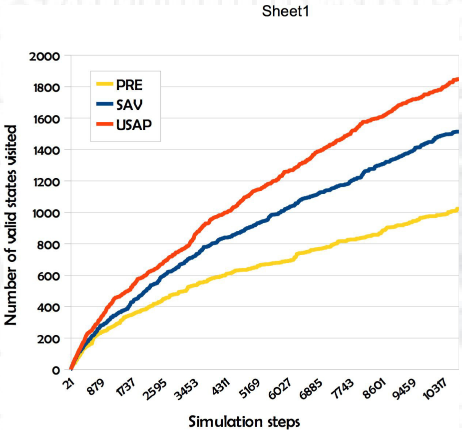


Figure : Hose transportation task with GEDS model: on-line predictive performance. Number of valid states visited. Action selection policies: PRE random selection, SAV state action vetoes, USAP undesired state-action prediction.

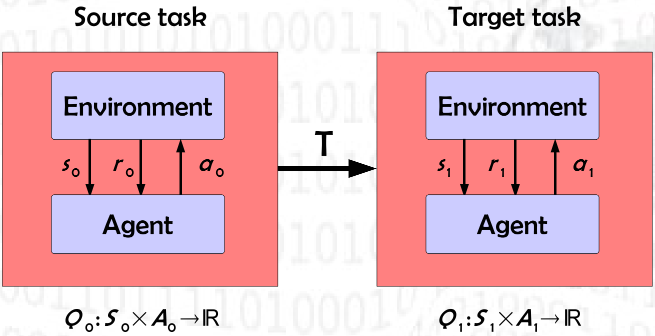
Contents

Transfer Learning

- System complexitity $- > +$ time needed to learn
  - Hose GEDS model in Matlab : 45 seconds to simulate a single step with 2 robots
- Transfer Learning,[3] transfers knowledge acquired in training on a simplified task to the full-fledged target task
  - Simplified version of the hose transportation task that used line segments to represent the hose

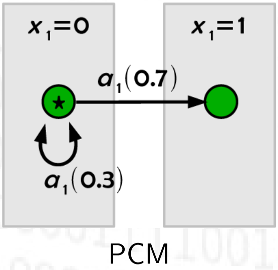
[3] *Borja Fernandez-Gauna, Jose Manuel Lopez-Guede, Manuel Graña; Transfer Learning with Partially Constrained Models: application to reinforcement learning of linked multicomponent robot system control; Robotic and Autonomous Systems, 61(7) (2013):694–703*

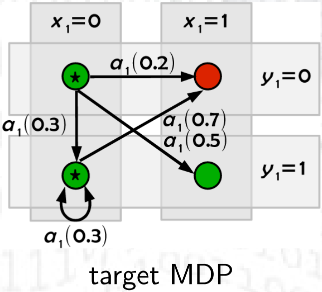
Trasfer learning



Transfer Learning with Partially Constrained Models

- Partially Constrained Model (PCM) : removing (by aggregation) state variables related to constraints
  - hand made simplifications
  - Knowledge transfer: Q-table





Transfer Learning

Definition

Source $M^s =< S_s, A, P_s, R_s >$ and a target $M^t =< S_t, A, P_t, R_t >$ MDPs, M^s is a PCM of M^t if

1. P1: $S_t = S_s \times S_Y$, where S_Y is state space of variables Y removed.
2. P2: Transition probability mass preservation:
$$\sum_{[t]_{S_s}=[s']_{S_s}} P_t(s,a,t) = P_s\left([s]_{S_s}, a, [s']_{S_s}\right)$$
3. P3: Positive reward function preservation
$\forall s \in S;\ R_t(s) \geq 0 \Rightarrow R_t(s) = R_s\left([s]_{S_s}\right)$.
4. P4: Negative rewards almost preservation
$\forall s \in S;\ R_t(s) < 0 \Rightarrow \left(\left[R_t(s) = R_s\left([s]_{S_s}\right)\right] \vee \left[R_s\left([s]_{S_s}\right) = 0\right]\right)$.

Transfer learning

- Initialize the Q-Matrix of the target task ($Q_t(s,a)$) with the Q-values learnt from the source task ($Q_s(s,a)$):

$$Q_t(s,a) = Q_s\left([s]_{S_s}, a\right), \tag{2}$$

- The effective action repertoires are likewise mapped:

$$A_t^e(s) = A_s^e\left([s]_{S_s}\right), \tag{3}$$

where A_s^e and A_t^e are source and target repertoires.

Transfer learning

Theorem

For all states $s \in S_t$, the effective action repertoires in the target MDP will be a subset of the effective action repertoires in the projected state in the PCM:

$$A_t^e(s) \subseteq A_s^e\left([s]_{S_s}\right).$$

Transfer Learning

Theorem

(No state value degradation in transfer) Given PCM optimal $Q_s^*(s,a)$ *values and* $A_s^e(s)$ *sets. Greedy source action selection* $\pi_t^g(s) = \arg\max_{a \in A_t^e(s)} Q_s^*\left([s]_{S_s}, a\right)$ *in* M^t *is an upper bound for the optimal state values in the target task, i.e.* $V_t^{\pi_t^g}(s) \geq V_t^*(s)$.

Transfer Learning

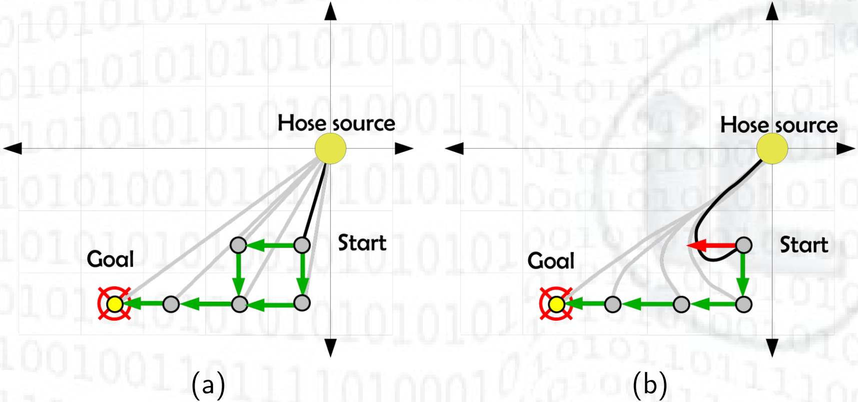
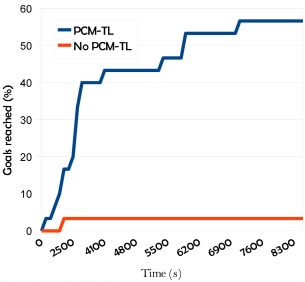
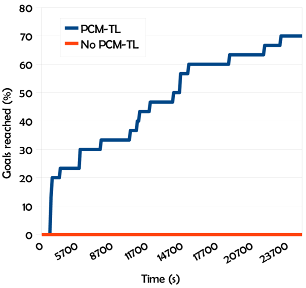


Figure : An example of the differences regarding constraints in the hose transportation problem: (a) Simplified PCM and (b) GEDS simulation environment.

Transfer Learning with Partially Constrained Models

- Succesful runs with 3 and 4 robots





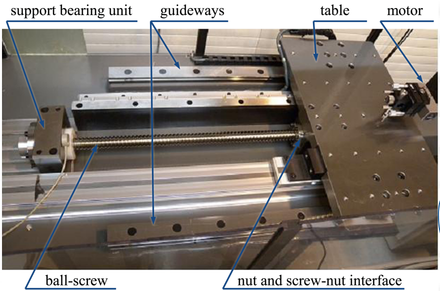
Contents

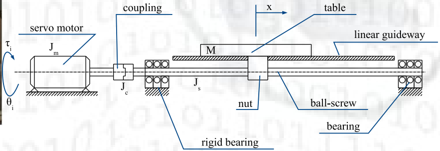
Continuous Action and State spaces

- Most control systems present continuous actions and state variables
- Q-Learning need discrete sets from continuous-valued actions and states
  - this does not always suffice for an accurate control system
  - the size of the table grows exponentially
- A better approach is to use approximate the value function (Q or V) using a Value Function Approximation

Continuous Action and State Spaces

Example application to control a ball screew feed drive[4]

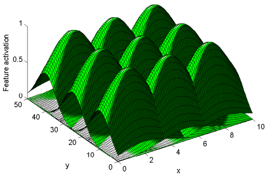


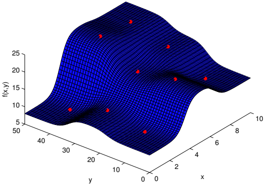


[4]Borja Fernández-Gauna; Igor Ansoategui; Ismael Etxeberria-Agiriano; Manuel Graña Reinforcement Learning of ball screw feed drive controllers Engineering Applications of Artificial Intelligence Volume 30, April 2014, Pages 107–117

Value Function Approximation

- An example: a 2-input/1-output function approximated with a network of Gaussian Radial Basis Functions





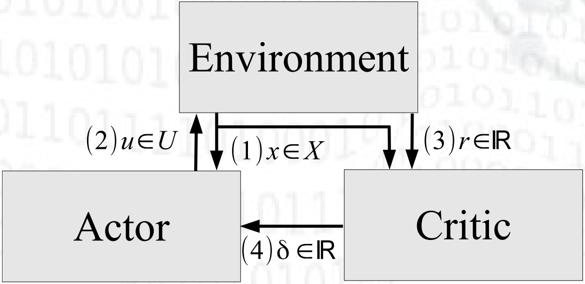
- On the left, the activation functions for each feature
- On the right, the approximated function $\hat{f}(x,y) = \sum_i \sum_j \theta_{i,j} \phi_{i,j}(x)$

Actor-Critic

- The actor selects and executes a control action
- The critic receives a reward assessing how desirable the last action was and gives a policy correction to the actor

$$\delta_t = r_t + \gamma * \hat{V}(s_t) - \hat{V}(s_{t-1})$$

.



Actor-Critic algorithms

- Q-AC: the actor implements Q-function with discrete action space, the actor executes an action a in state s, receives the TD error from the critic, and updates the $\hat{Q}(s,a)$ estimation:

$$\theta_t^Q \leftarrow \theta_{t-1}^Q + \alpha_t \cdot \delta_t \cdot (min + (1 - \pi(s,a))) \cdot \frac{\partial \hat{Q}_{t-1}(s_{t-1}, a_{t-1})}{\partial \theta_{t-1}^Q}, \quad (4)$$

.

- Policy gradient Actor-Critic (PG-AC): actor implements a continuous valued policy $\pi_a(s)$:

$$\theta_t^a(s) \leftarrow \theta_t^a(s) + \alpha_t \cdot \delta_t \cdot (a_t - \pi_a(s)) \cdot \frac{\partial \pi_a(s_{t-1})}{\partial \theta_{t-1}^\pi}, \quad (5)$$

- Continuous Action-Critic Learning Automaton (CACLA). The actor only updates its policy if the critic is positive,:

$$\textit{if } \delta_t > 0: \qquad \theta_t^a(s) \leftarrow \theta_t^a(s) + \alpha_t \cdot (a_t - \pi_a(s)) \cdot \frac{\partial \pi_a(s_{t-1})}{\partial \theta_{t-1}^\pi}. \quad (6)$$

Actor-critic

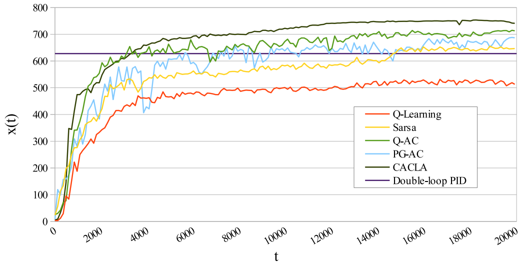
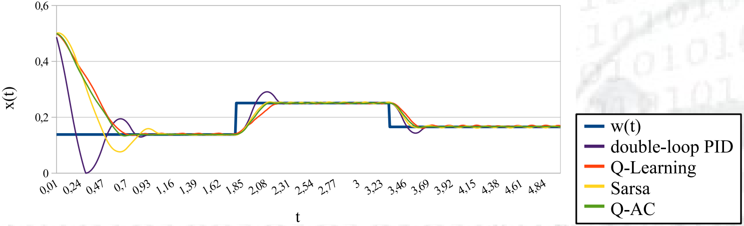
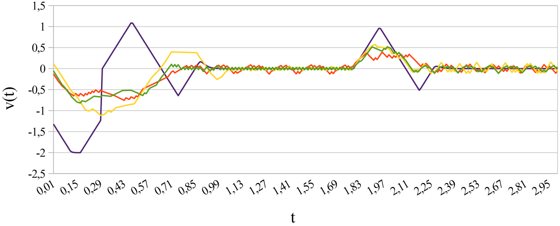


Figure : Evaluation of the controllers in Experiment A: average discounted rewards. PID controller has constant reward

Actor-critic





Contents

MARL

- Many real situations can not be modeled by a single agent
    - Multicomponent Robotic Systems:
    - Power distribution systems
    - Intelligent trasportation systems
- MARL tries to make manageable the complexity of multi-agent system control
    - Decomposition into concurrent learning processes
    - Synchronous vs. asynchronous decision making processes

MARL

- Two basic views of RL in Multiagent Systems:
  - Agents are unaware of the actions taken by other agents
    - Agents don't know what actions other agents choose
    - No communication required, but convergence can only be guaranteed under strict conditions
  - Agents aware of the actions taken by other agents
    - Agents know what actions are choosen by other agents
    - Communication required, stronger guarantees of convergence

Challenges

- Agents need to coordinate either explicitly or implicitly:
  - Learning while other agents are also learning and changing their policies
- State and action space decomposition
- Joint action composition
- Formal proofs of convergence are difficult and scarce
  - Non-stationary MDP (agents are learning and changing policies)
  - Problems are modeled as Stochastic Games

Contents

Stochastic Games

- MDP become Stochastic Games in MAS
- Stochastic Games are defined by a tuple $\langle S, \mathbf{A}, P, \mathbf{R} \rangle$, where
  - The set of joint-actions is $\mathbf{A} = \bigcup_{i=1}^{n} A_i$
  - Each agent receives a possibly different reward $\mathbf{R}(s) = \{R_1(s)\ R_2(s) \ldots R_n(s)\}$
    - In control tasks, Cooperative SG, where $R_1(s) = R_2(s) = \ldots = R_n(s)$
    - In competitive settings, optimal policies lead to Nash equilibria?

Team Q-Learning

- Naive MARL algorithm: Team Q-Learning
  - Multi-agent extension of single-agent Q-Learning
  - Each i-th agent stores its local estimation of the global state-action value function $Q^i(s, \mathbf{a})$, where $\mathbf{a} \in \mathbf{A}$
  - The size of this table becomes $|S| \cdot |\mathbf{A}|$
    - Assuming that all agents have the same set of local actions A to choose from: $|S| \cdot |A|^n$

$$Q_t^i(s, \mathbf{a}) = (1-\alpha)\, Q_{t-1}^i(s, \mathbf{a}) + \alpha \cdot \left[r + \gamma \cdot \arg\max_{\mathbf{a}'} Q_{t-1}^i(s', \mathbf{a}') \right]$$

Contents

Distributed Value function

- One of the earliest MARL proposals[5] as distributed RL (DRL)
- A hierarchy of distributed information and learning processes
  - Diverse degrees of communication between agents
  - Diverse degrees of global information
- Variations of Bellman equation:

$$V(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p\left(s' \middle| s, a\right) V\left(s'\right) \right\}$$

$$V^*(s) = \left\langle \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right\rangle$$

[5] J. Schneider, W.-K. Wong, A. Moore and M. Riedmiller "Distributed value functions" Proc. Int. Conf. Mach. Learn. 1999, pp. 371-378,

- Global reward DRL

$$V_i(s) = \max_{a \in A_i} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \right\}$$

- Local reward DRL (no communication)

$$V_i(s) = \max_{a \in A_i} \left\{ R_i(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \right\}$$

- Distributed reward DRL (communication of rewards with neighbors)

$$V_i(s) = \max_{a \in A_i} \left\{ \sum_j f(i,j) R_j(s,a_j) + \gamma \sum_{s' \in S} p(s'|s,a) V_i(s') \right\}$$

- Distributed value function DRL (communication of value functions with neighbors)

$$V_i(s) = \max_{a \in A_i} \left\{ R_i(s,a) + \sum_j f(i,j) \gamma \sum_{s' \in S} p(s'|s,a) V_j(s') \right\}$$

Distributed Value Functions

- Distributed state and reward Q-learning for DVF

$$Q_t^i(s_i,a_i)=(1-\alpha)\,Q_{t-1}^i(s_i,a_i)+\alpha\cdot\left[R_i(s_i,a_i)+\gamma\cdot\sum_j f(i,j)\max_{a_j'} Q_{t-1}^j(s_j',a_j')\right]$$

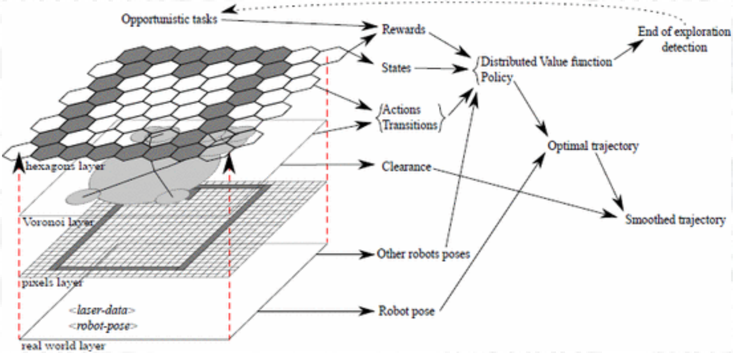
Multirobot exploration

- Multirobot exploration[6]
  - Minimize overlapping of sensor span
  - Maximize joint coverage
  - Robots need only to communicate when/with physically near
- Distributed state common reward (coverage)

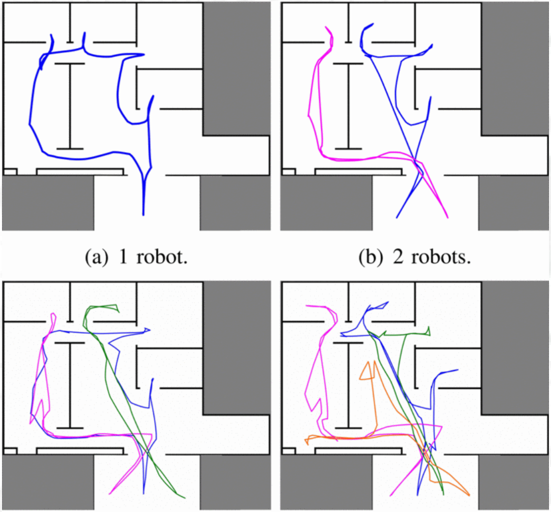
$$\forall s_i \in S; V(s_i) = R_{explo}(s_i) + \gamma \max_{a_i \in A} \sum_{s' \in S} T(s_i, a_i, s') \left[V_i(s') - \sum_{j \neq i} f_{ij} P_r(s'|s_j) \widehat{V_j(s')} \right]$$

[6] Matignon, Laëtitia; Jeanpierre, Laurent; Mouaddib, Abdel-Illa, Distributed value functions for multi-robot exploration, ICRA 2012, pp.1544 - 1550; doi 10.1109/ICRA.2012.6224937

Multirobot exploration



Multirobot exploration



(a) 1 robot. (b) 2 robots.

(c) 3 robots. (d) 4 robots.

Smart Grid

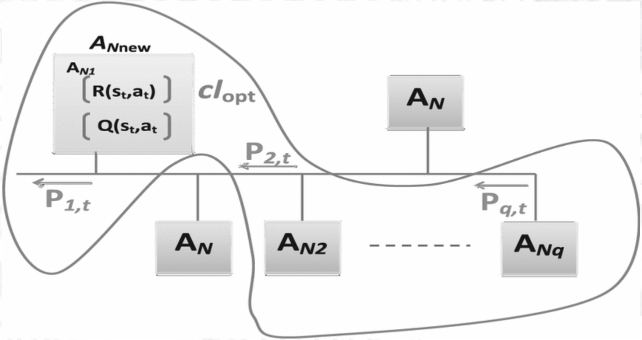
- Renewable energy sources (wind, sun, ...) are random
    - power flows reverse direction according to environmental conditions
- Smart Grid tries to falance their contributions to obtain stagy power supply
- Modelling as Multiagent system (MAS)[7]
    - Managed by a Plug and Play (PnP) algorithm
        - interoperable model and information system
        - orderly connection and disconnection
        - minimize disturbances to the supply-and-demand balance
    - The role of VDF: online adjustment of power contribution/consumption per active node

[7] Shirzeh, H.; Naghdy, F.; Ciufo, P.; Ros, M., Balancing Energy in the Smart Grid Using Distributed Value Function (DVF), Smart Grid, IEEE Transactions on, march 2015, doi 10.1109/TSG.2014.2363844

Smart Grid

- Operation of the MAS PnP when a new node is added
  - Cluster formation by dialog with the central controller, maximizing an index of normalized costs, distance, and capability

$$U_{\text{new,p}} = \sum_{k=1}^{p} \mathrm{NN}_{\text{new},k}^{Co} + \sum_{k=1}^{p} \mathrm{NN}_{\text{new},k}^{Ca_t} + \sum_{k=1}^{p} \mathrm{NN}_{\text{new},k}^{Di} + \sum_{k=1}^{p} N_k^{Av_t}.$$



Smart Grid

- Load balance with DVF
  - Reward within cluster of source/drain nodes

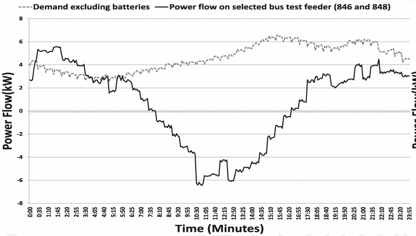
$$\text{Power deviation index} = \sum_{i=1}^{q} (P_{i,t} - P_{i,t-1})^2$$

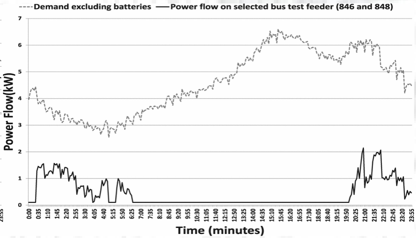
  - Q-learning

$$\begin{aligned} Q_{\text{new}}(s_t, a_t) =& (1-\alpha) Q_{\text{new}}(s_t, a_t) \\ & + \alpha \left[R_{\text{new}}(s_t, a_t) + \sum_{i \in \text{Neigh(new)}} f(\text{new}, i) V_i\left(s_i'\right) \right] \\ \text{where, } V_i\left(s_i'\right) =& \max_{a \in A_i} Q_i\left(s_i', a\right). \end{aligned}$$

Smart Grid

Without and with PnP algorithm in example topology





Contents

Distributed Round-Robin Q-Learning

- Distributed Round-Robin Q-Learning (D-RR-QL)[8] is a two-phase learning algorithm
  - First, agents take actions sequentially following a round-robin execution schedule
    - Local actions can be vetoed using MSAV without interference of the rest of agents
  - Secondly, a message-passing scheme is used to coordinate the agents and approximate the optimal joint-policy
- D-RR-QL allow veto state-action pairs (MSAV) efficiently in distributed RL scenarios

[8] Borja Fernandez-Gauna; Ismael Etxeberria-Agiriano; Manuel Graña Learning Multirobot Hose Transportation and Deployment by Distributed Round-Robin Q-Learning PlosOne, Volume 10(7): e0127129; DOI 10.1371/journal.pone.0127129

Distributed Round-Robin Q-Learning

Definition

A *Cooperative Round-Robin Stochastic Game* (C-RR-SG) is a tuple $< S, A_1 \dots A_N, P, R, \delta >$, where

- N is the number of agents.
- S is the set of states, fully observable by all the agents.
- A_i, $i = 1, \dots, N$ local actions i-th agent.
- $P : S \times \cup A_i \times S \to [0,1]$, $i = 1, \dots, N$ is the state transition function $P_t(s, a, s')$ that defines the probability of observing s' after agent $\delta(t)$ executes, at time t, action a from its local action repertoire $A_{\delta(t)}$.
- $R : S \times \cup A_i \times S \to \mathbb{R}$ is the shared scalar reward signal $R_t(s, a, s')$ received by all agents after executing a local a action from $A_{\delta(t)}$.
- $\delta : \mathbb{R} \to \{1, \dots, N\}$ is the cyclic turn function implementing the Round-Robin cycle of agent calling for action execution.

Distributed Round-Robin Q-Learning

The Bellman equation for a joint policy π in a C-RR-SG is

$$\begin{aligned} V^{\pi}(s,i) &= E^{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right\} \\ &= \cdot\sum_{a\in A_i}\pi_i(s,a)\sum_{s'}P(s,a,s')\left[R(s,a,s')+\gamma V^{\pi}(s',i+1)\right], \end{aligned}$$

The state-action value function for agent i following joint policy π can be expressed as

$$Q^{\pi}(s,a,i)=\sum_{s'}P(s,a,s')\left[R(s,a,s')+\gamma V^{\pi}(s',i+1)\right] \qquad (7)$$

Distributed Round-Robin Q-Learning

Communication free D-RR-QL:

- each agent has a local Q-table updated at the end of an RR cycle
- using the information of the rewards along the cycle broadcasted to all agents:

$$\begin{aligned} Q_t^i(s,a) &= (1-\alpha_t)\, Q_{t-N}^i(s,a) \\ &+ \alpha_t \left[\sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q_t^i\left(s_{t+N}, a'\right) \right] \end{aligned}$$

applied when $s_t = s, a_t = a, \delta(t) = \delta(t-N) = i$.

- no the need to know the Q-tables of other agents.

Distributed Round-Robin Q-Learning

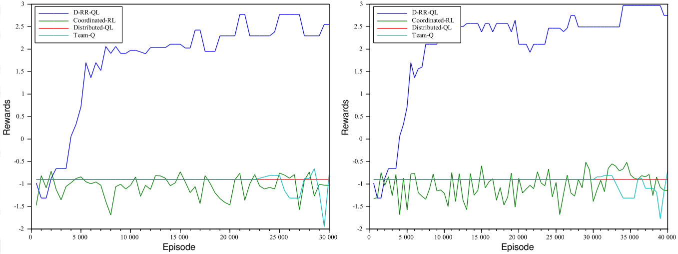
Theorem

Convergence of the communication-free D-RR-QL to the optimal policy, $Q_t^i(s,a) \rightarrow Q^*(s,a,i)$ as $t \rightarrow \infty$, for a given a C-RR-SG $\langle S, A_1 \ldots A_N, P, R, \delta \rangle$ is guaranteed when each agent fulfills the conditions of convergence of single-agent Q-Learning in a MDP.

Joint action constructed by a message passing algorithm and greedy selection at each agent.

Distributed Round-Robin Q-Learning

- D-RR-QL with MSAV vs. Coordinated-RL, Distributed-QL and Team-QL



Contents

Future research

- Most of the cooperative MARL literature is:
  - based on Q-Learning approaches
    - cannot deal with continuous state-action spaces
  - challenges addressed so far
    - solving coordination issues
    - dealing with the uncertainty of the other agents' changing policies

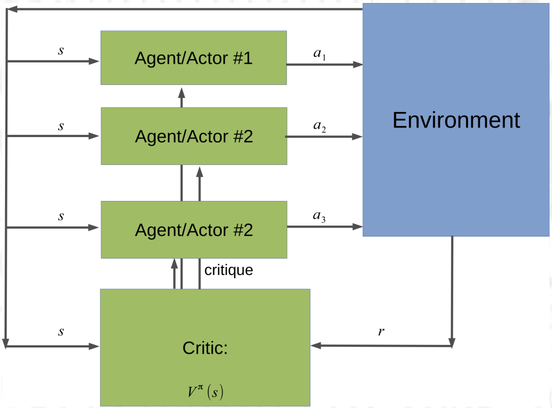
Future research

- if we assume
  - Homogeneous agent systems
  - That the learning parameters are shared and communicated to all the agents?
    - this is easier than communicating rewards, actions or states
    - communication requirements can be reduced using consensus-based mechanisms
  - A central observer in charge of learning the value of the joint policy?
    - this might be more assumable than a centralized agent

Future research

- We propose a multi-agent implementation of Actor-Critic methods
  - each agent implements a policy (**actors**)
  - a centralized observer learns the joint policy's value $V^{\pi}(s)$ (the **critic**)
- This would allow
  - continuous states and actions
  - VFAs to represent the policies and the value function
- **Actors** can improve their policies locally according to global **critic's** feedback that evaluates the joint performance

Multi-agent Actor-Critic



Contents

Conclusions (Pro)

- RL methods offer a promising alternative to traditional control strategies
  - Little input from the designer
  - No need of a precise dynamic model
  - Autonomous learning
  - Inherently adaptive methods
- MARL is the natural extension of RL to multi-component control
  - Problem complexity reduction by decomposition

Conclusions (Challenges)

- MARL realtime operation
  - True decentralized/distributed learning
    - Convergence is not assured in very general settings
    - Convergence is very slow
  - Toy problems: simulations
  - Generalization to multi-agent actor-critic
  - Exploration vs. exploitation <=>
    - distributed concept drift detection
    - non-stationary regime detection