On Collective Reinforcement Learning

Techniques, Challenges, and Opportunities

Gianluca Aguzzi   Mirko Viroli
gianluca.aguzzi@unibo.it   mirko.viroli@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

2021/12/09

Introduction

Motivation

- Learning is a key aspect to drive *adaptivity* (by reason under uncertainty)
- Intelligent agents improve their performance with *experience*
- We would like to improve adaptiveness with raw experience and, hence
- We would like to bring Reinforcement Learning methodology in such kind of systems

Lecture goals

- Understading the challenges related to the Multi-Agent System (MAS) domain
- Show Patterns and Architecture applied in Collective Systems
- Hands-on in some pratictal example of Collective Learning

Single-Agent Learning

What have you seen so far ...

- Markov Decision Process (MDP) to model the agent-environment interactions
- Find learning process that eventually lead to an *optimal* policy π^*
- Q-Learning (in general *value-based approaches*) as a prominient algorithm case to reach converge

... But this works only under some conditions

- Reward hypothetesis
- Full environment observability and Markovian property
- Stationary environment
- State/Action space should be small enough to be stored in-memory (otherwise, we should leverage function aproximators)

Partial Observable environments

Definition

Agent does not have a *perfect* and *complete* knowledge perception of the state of the environment

They are quite typical in (non-)complex systems

- Card-games (poker, black-jack, . . .) — Why?
- A driving card
- Swarm of drones

Partial Observable Environments

Partial Observable MDP $\mathcal{P}$ (POMDP)

- Agent cannot directly observe the system state, but he can make an *observation* that depend on this state
- POMDP is a tuple (S, A, T, R, Ω, O)
- S, A, T, R are the same variable described in MDP
- Ω is the set of observation perceive by the agent $\{o_1, \ldots, o_n\}$
- O is the set of conditional probability $O(o|s, a)$
- I want to learn a policy that depends on o but maximise a reward function that depends on s
- The agents need to build a belif state from the observations history

Non-stationary environments

Definition

The environment model (e.g. the random variable associated with it) change over the time.

MDP are Stationary by definition . . .

- . . . But, real-case environment dynamics could change over time (e.g. markets, city trafic, routing networks)
- Pratically, it seems that RL works well even in this case (online learning)
- *But, we loose converge guarantee.*

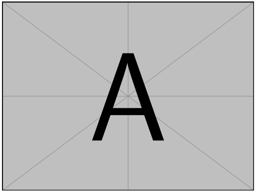
From Single-Agent To Multi-Agent

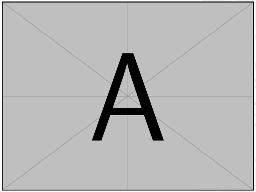
Multi Agent Reinforcement Learning (MARL)

*Multiple agents learn **togheter** the best a policy that maximise a long term reward signal.*

Considerations

- If multiple agent exist, but only **one** agent learn by experience, then it is a single agent problem (e.g. single player videogames)
- So, MAS + Learning $\not\Longrightarrow$ MARL, **but** MARL $\Longrightarrow$ MAS





Stochastic Game $\mathcal{S}$ (or Markov games)

- Extension of MDP to the MAS regime
- Common abstraction in MARL algorithm

Definition

- $\mathcal{S}$ is a tuple $< N, S, \{A^i\}_{i \in \{1,\ldots,N\}}, P, \{R^i\}_{i \in \{1,\ldots,N\}} >$
- N is the number of agents ($|N| > 1$)
- S is the global environment state
- A^i is the action state of agent i. $\mathbb{A} := A^1 \times \cdots \times A^N$ is the joint action space
- $P : Sx\mathbb{A} \rightarrow \mathcal{P}(S)$ is the state transaction. $\mathcal{P}$ is a discrete probabilistic distribution (associate for each $s \in S$ a probability)
- $R^i : S \times \mathbb{A} \times S \rightarrow \mathbb{R}$ is the reward signal
- Typical time evolution: $S_0, \mathbb{A}_0, \mathbb{R}_1, S_1, \ldots$

Stochastic games: Example

Paper Rock Scissor

- $N = 2$
- $A^1 = A^2 = \{\text{Paper, Rock, Scissor}\}$
- $S = \{\ \}$
- $R^1 = R^2 = \begin{array}{c} \\ \textit{Rock} \\ \textit{Paper} \\ \textit{Scissor} \end{array} \begin{array}{c} \begin{array}{ccc} \textit{Rock} & \textit{Paper} & \textit{Scissor} \end{array} \\ \begin{bmatrix} 0,0 & -1,1 & 1,-1 \\ 1,-1 & 0,0 & -1,1 \\ -1,1 & 1,-1 & 0,0 \end{bmatrix} \end{array}$

MARL Systems: Task type

Cooperative

- Agents share the same reward function ($R^1 = \cdots = R^N$) in order to accomplish a collective goal

Competitive

- Agents compete with each other to maximise a long term return
- Board Games, Video games

Mixed

- Agent can both compete and cooperate in order to maximise a global reward function
- Also called *General Sum games*

On Cooperative Task

Homogeneous

- Each agent has the same capability ($A^1 = \cdots = A^N$)
- The overall goal is to find the best policy that is the same for each agent ($\pi^* = {\pi^*}_1 \ldots {\pi^*}_N$)

Heterogenoues

- Each agent could have different capabilities (in the worst case, $A^1 \neq \ldots \neq A^N$)
- Each agent has its local policy that should be maximised following the global collective goal

MARL Systems: Learning Scheme

Cetralised Learning and Centralised Execution

- *One* agent with a global view of the system (in the cloud? in a server?)
- Node send their perception to that Node
- With them, it create a global state of the system
- With the current policy, it chooses the action that nodes should performance and send to them (*Control*)
- In the next time step, it evaluates the reward function and update the policy accordingly (*Learn*)
- *Are the nodes agents?*
- Both used in offline and pure-online setting

MARL Systems: Learning Scheme

Decentralised Learning and Distributed Execution

- Each nodes has their local policy/value table
- They can perceive the environment state (or can observe a part of it)
- With the current state, they performance an action (*Control*)
- In the next time step, they update their policy following a local reward function
- Both used in offline and pure-online setting

MARL Systems: Learning Scheme

Centralised Learning and Distributed Execution

- A offline-learning online execution patterns
- *Simulation time*
  - Each node follow the typical $o_t, a_t, o_{t+1}, a_{t+1}, \ldots$ trajectory using a local policy
  - After an episode, this trajectory (or something dirived from it) will be sent to a central learner
  - It, using a global view of the system, improve the policies of the agents
  - At the end of the traning phase, the policy will be shared to the agent s
- *Execution time*
  - Each agent has the local policy distilled during the simulation time
  - With it, they act in the environment
- A semplific description, more elaborate Techniques exists (i.e. agents share the gradient to the central leaner)

Learning in Collective Adaptive System *

- The collective goal could be accomplished through competition and/or cooperation
- The system could be heterogenoues or homogeneous
- The agent numers is not bounded (openness)
- Distributed control – i.e. no central authority exists

*Mirko D'Angelo et al. "On learning in collective self-adaptive systems: state of practice and a 3D framework". In: ACM, 2019, pp. 13–24. url: https://doi.org/10.1109/SEAMS.2019.00012

Learning in CAS: Challenges

CASs are partial observable

Each agent could only perceive a part of the system through its sensors.

Learning in CASs make the environments non-stationary

Each agent learns concurrently $\implies$ by the eye of the other agent the environment is in continuous changes.

Course of dimensionality – MAS combinatorial nature

When we have to deal with a large number of agents, the overall state-action space is increasing exponentially to the number of agents — so a central learner cannot solve the learning problem.

Learning in CAS: Challenges

Multi-Agent credit assigment problem

Tipycally, in CAS, a global reward function exisist. But it is hard to understand the influence of a single agent to the overall collective behaviour.

A lack of a global clock

CASs are distributed systems $\implies$ a global synchronization clock does not exist, making the standard Stachocastic game model quinte inaquate.

Sample efficency

Action-space and state-space are very large in CASs $\implies$ the problem of sample efficiency (i.e. how many samples does the RL need to reach a good policy?) arise as in the Deep Learning context.

On sample efficency: Example of learning time

A Nowadays problem..

- Nowadays, Deep Learning Techniques are used to train complex neural networks
- When applied to Reinforcement Learning, the traning phase requires millions of interactions for an agent to learn
- In the Multi Agent Setting is even worst..
- In [Jad+18] they train 30 agents in 450k games!

On scalability: Single Agent Learner Example

Learning setting

- A central agent (i.e. in the cloud? In a server?) see the entire system
- Standard RL algorithm (tabular) create a table with the size of $|S \times A|$
- But the system-wide action space is the cartesian product of the agent action-space, so the A cardinality is $|A|^N$
- With 100 agents, we already reach an action space with more action than the particle in the universe.

Learning in CAS: Our today focus

Constraints

- Learning in cooperative systems: i.e. each entity share the same reward fuctions
- Learning in homogeneous systems: i.e. each entity is interchangable with each other
- We consider partial observability not as a core problem

Homogenous system: Implications

- The optimal policy is the same for the whole system
- During the learning, the system can use different policy (e.g. to reduce sample efficency)
- We reduce the action space
- We reduce the non-stationarity problem

Learning in CAS: Models

Dec-POMDP

- Extension of POMDP to Multi Agent settings
- N agent act in a Markovian environment *but* they perceive only partial information about it (observations)
- ! Does not consider homogeneity

SwarMDP †

- Consider an homogeneous population of agents (i.e. same action, observation space and same policy)
- Learning lead to single policy that map observation (not history) to action
- ! Time continuinig to be synchrnous

† Adrian Sosic et al. "Inverse Reinforcement Learning in Swarm Systems". In: *CoRR* abs/1602.05450 (2016). arXiv: 1602.05450. url: http://arxiv.org/abs/1602.05450

Learning in CAS: A simplified model

- Each agent is seen as a tuple $(M, \mathcal{N})$
- M is a local markov decision process (S is an observation space in common with the entire system)
- The global state system state is unknown
- Agents want to learn a policy $\pi(a|s, \mathcal{N})$ to share to the entire system
- When agent does not consider neighbours, we call the system as *Independent learners*
- When agent consider neighbours action to choose their local action, we call the system as *Joint action learners*

Learning in CAS: Can we use standard RL algorithm?

In a centralised learning decentralised execution settings

- Each agent explore the environment with the same policy ϕ
- A central learners improve the policy following the agents perception
- When a *good* policy is found, it is deployed in a real system
- Then, the central learner it is not require anymore

Ideally yes . . .

- . . . But this allow only offline learning
- This pratically lead to lazy agent – the exploration part is limited
- Does not consider other agent actions – each agent act independently from each other

Independent Learners

Configuration

- Each agent learn concurrently an optimal local policy
- This lead to the global optimal policies as an emergent behaviour
- The homogeneity is driven by the same reward function and action/observation spaces
- More agent lead to a more exploration
- Not converge to global optimum, but good pratical performance are founded in various works: [TA07; TAW02; WS06]

Independent Learners

Implications

- *Scalable*: the learning algorithm does not depend to the system size
- *Easy to implement*: the algorithm are the same developed for single agent context
- *Offline and Online*: Can be used both at simulation time or at runtime
- *Increase non-stationarity*: the environment dynamics change continously as the agent learn new policy

Decentralised Q-Learning[‡]

Idea

Each agent is a Q-Learner and does consider other agents as a part of the environment

Considerations

- 👍 Simplest extension of Q-Learning into the multi agent domain
- 👎 Need to use Greedy in the Limit of Infinite Exploration (GLIE) policy to reach good performance
- 👎 Complex and aplication dependant parameter tuning

[‡] Ming Tan. "Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents". In: ed. by Paul E. Utgoff. Morgan Kaufmann, 1993, pp. 330–337. doi: 10.1016/b978-1-55860-307-3.50049-6. url: https://doi.org/10.1016/b978-1-55860-307-3.50049-6

Hysteretic Q-Learning[§]

Idea

- Decentralised Q-Learning does not take in consideration non-stationarity problem
- An idea could be to apply a *correction* factor that reduce the non-stationarity born for the concurrent learning
- Hysteretic Q-Learning uses an *hysteresis* heurestic to handle this problem
- It suppose an optimistic behaviour, giving more weight to good action then the bad action (*due to the exploration of nearby agents*)

[§] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. "Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams". In: IEEE, 2007, pp. 64–69. url: https://doi.org/10.1109/IROS.2007.4399095

Hysteretic Q-Learning

Implementation

- Use two learning factor, α and β, with $\alpha > \beta$
- The Q update is evaluated to consider the goodness of an action/space pair: $\delta(s, a) = r_t + \gamma * argmax_{a'} Q(s', a') - Q(s_t, a_t)$
- When $\delta > 0$ it means that we improve our experience, so we use α as learning rate, β otherwise
- The Q update became:

$$Q(s_t, a_t) = \begin{cases} Q(s_t, a_t) + \alpha\delta(s_t, a_t) & \text{if}(\delta(s_t, a_t)) > 0 \\ Q(s_t, a_t) + \beta\delta(s_t, a_t) & \text{otherwise} \end{cases}$$

Hysteretic Q-Learning

Idea

- 👍 It improve standard Q-Learning performance without adding more complexity
- 👍 It is currently used in Deep Reinforcement Learning to handle complex System
- 👎 It is an heurestic, so it does not have any theoretical guarantes
- Other extension follow this idea and suffer from the same pros and cons. The most famous are Distributed Q-Learning, Lenient Q-Learning, Win-Or-Learn-fast

QD-Learning[¶]

Idea

- Each agent know it local actions and can communicate only with a restricted neighbourhood
- The Q update depends on the Q values of the neighbours
- To do that, the system mantain the Q table of different time step.
- Q are indexed with: Q_t^i (in english: the Q table of the agent i at the time step t)
- It uses *innovation* and *consensus*

[¶]Soummya Kar, José M. F. Moura, and H. Vincent Poor. "QD-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through *Consensus + Innovations*". In: *IEEE Trans. Signal Process.* 61.7 (2013), pp. 1848–1862. url: https://doi.org/10.1109/TSP.2013.2241057

QD-Learning

Update rule

- $consensus(s_t, a_t) = \sum_{i \in \mathcal{N}} (Q^{me}_t(s_t, a_t) - Q^i_t(s_t, a_t))$
- $innovation(s_t, a_t) = (r_t + \gamma * argmax_{a'} Q^{me}_t(s', a') - Q^m e_t(s_t, a_t))$
- *innovation* is the standard Q advantage evaluation
- Finally, each agent update their table as:
  $Q^{me}_{t+1}(s_t, a_t) = Q^{me}_t - \beta * consensus(s_t, a_t) + \alpha * innovation(s_t, a_t)$

Discussion

- It is shown that it converge asymptotically to an optimal policy ...
- ...under constraints of networks connection
- ...with full state observability
- ...with specific constraints on α and β
- ...with a predefined neighbourhood graph

Joint Action Learners

From Independent to Joint actions

- Independent learners tend to make the environment highly non-stationarity
- Futhermore, they do not consider other agents $\leftarrow$ collective behaviour through emergence (at exception of QD-learning)
- Ideally, we can reduce non-stationarity and we want to consider other agents by creating policy that observe other agents actions (tipycally denoted by: $\pi_{i,\mathcal{N}}$))
- This does not scale with the agent population.
- A possibility to reduce the space is to consider only the neighbours (in a tipycally CAS settings)

Neighbourhood Q-Learning[‖]

- Q-Learning mixed with Joint-action spaces
- Each agent improve a Q-function consider also the action taken by the neighbourhood ($a_{\mathcal{N}}$)
- A way to handle that, it to consider the state as $k_t = (s, a_{\mathcal{N}})$
- $Q(k_t, a) = Q(s, a_{-1}) + \alpha * (r_t + \gamma * argmax_{a'} Q(k', a'))$

Implications

- 👍 Consider neighbourhood actions $\implies$ reduce non-stationarity problem
- 👎 Q depends on agent neighbourhood $\implies$ the Q table size increase *exponentially* with the neighbourhood

[‖] Alexander Zai and Brandon Brown. *Deep reinforcement learning in action*. Manning Publications, 2020

"Just" a scartch to the surface :)

A recap

- The approaches handle only partially the collective learning problem (scalability, cooperative, non-stationarity)
- They are used nowadays but in combination with other approaches (neural networks, reply buffer, differential reward) ...
- ...But today no a theretical solution exists yet for large scale collective learning
- Note: we do not conver the "game theoretical" background, current main interest in literature (*problem* consider few agents)

State of the art: on advanced topics

Deep Learning as a mechanism to solve complex tasks

- Nowadays, Deep Learning architecture are typically used in MARL
- For handling non-stationarity, *actor-critic* methodology are used
- For handling partial-observability, *recurrent* networks are applied
- For handling large-state/action space, *deep architecture* are considered
- are all *heurestic* $\implies$ some approach could does not work in different aplication

On Actor-Critic: Policy Gradient Methods

- What we see so far are *value-based* methods $\implies$ learn value function that guide the agent's policy
- Why we cannot learn the policy *directly*?
- In some case, the value function could be very complex but not the policy
- Futhermore, value-based method lead to determistic policy, but in some case only a *stochastic* policy is the best one (e.g. in Rock paper scissor)

On Actor-Critic: Policy Gradient Methods

Idea

- Optimizing the policy ($\pi(a|s, \theta)$) w.r.t the exceptected return by gradient descent (or ascent).
- $\pi(a|s, \theta)$ is read as: the probability of the action a knowing the state s and the weight θ
- Generally, the loss function is described as: $J(\theta) = E \sum_{i=0}^{T} (\gamma * R_i)$
- The weight then, are updated as: $\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \implies$ standard Stachocastic gradient descent (ascent)

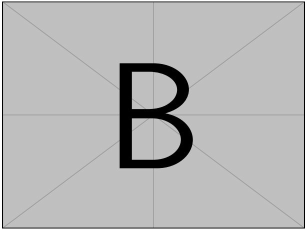
Considerations

- 👍 Convergence proof to local optimal
- 👎 Tends to have high variance (overfitting)

On Actor-Critic: Learning description

Idea

- Combine Policy Gradient and Value-Based Methods
- The Critic (the value function) is updated like standard value approaches (e.g. TD errors)
- The actor (the policy) is updated using the policy gradient update based on the critic estimation



On Actor-Critic: MARL settings

Idea

- The Actor are local policy, that take observation as input and return local action
- The Critic instead, is placed in a central server
- The Critic have a global view of the system $\implies$ it could use overall state information
- The Actor is then influenced with global data, but at *deployment* time it does not use it

Considerations

- 👍 A balanced between fully decentralised and centralised method
- 👎 Require simulations $\implies$ cannot be used in online systems

Handling Partial Observability

- As I mentioned, CAS are partially observable . . .
- . . . But the methods proposed does not handle this matter in any sense
- Recurrent Neural Network (i.e. Neural Network with *memory*) are an emergent trend to handle this issues
- In MAS settings, RNNs and Hysteretic update are used togheter to handle both non-stationarity and partial observability
- 👎 It is not clear if RNNs are really a solution for Partial Observability

Handling large scale systems

- Agent size was not a central problem in first MARL works
- Independent learners are typically deployed when we have to deal with several agent . . .
- . . . but it makes the learning process noising
- Lately, a novel trend consist in the Mean Field Reinforcement Learning

Handling large scale systems: On Mean Field Reinforcement Learning

- The learning problem is set up as a joint action learners
- In mean field, the system in model as two agent, the main agent and the rest
- In Mean Field Reinforcement Learning, the rest is computed as the mean action taken by the neighbourhood
- 👍 It has theoretical guarantee
- 👍 It is easily used also in CAS
- 👎 The approaches proposed are offline — no online adaptation.

Handling Multi-Agent credit assignment problem

- In CAS settings, typically we have a global reward function
- It is hard to understand the influence of an agent action to that signal
- This is know as Multi-Agent credit assignment problem
- It is tackle in COIN approach (i.e. differential reward) ...
- ... but is require to run several simulation to understand each agent influence
- COMA is a novel approach that uses a Actor-Critic setting with a contrafacutal baseline

Not explored issue: Global clock

- Works on Multi Agent Reinforcement Learning consider Stochastic games as the environments model
- There, agents action synchrnously, i.e. at the time step t+1 all agents have done their action
- This is impossible to assume in CAS
- Any ideas? Lamport logical clock?

On Collective Reinforcement Learning

Techniques, Challenges, and Opportunities

Gianluca Aguzzi   Mirko Viroli
gianluca.aguzzi@unibo.it   mirko.viroli@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

2021/12/09

References I

[DAn+19] Mirko D'Angelo et al. "On learning in collective self-adaptive systems: state of practice and a 3D framework". In: ACM, 2019, pp. 13–24. url: https://doi.org/10.1109/SEAMS.2019.00012.

[Jad+18] Max Jaderberg et al. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning". In: *CoRR* abs/1807.01281 (2018). arXiv: 1807.01281. url: http://arxiv.org/abs/1807.01281.

[KMP13] Soummya Kar, José M. F. Moura, and H. Vincent Poor. "QD-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through *Consensus + Innovations*". In: *IEEE Trans. Signal Process.* 61.7 (2013), pp. 1848–1862. url: https://doi.org/10.1109/TSP.2013.2241057.

References II

[MLL07] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. "Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams". In: IEEE, 2007, pp. 64–69. url: https://doi.org/10.1109/IROS.2007.4399095.

[Sos+16] Adrian Sosic et al. "Inverse Reinforcement Learning in Swarm Systems". In: *CoRR* abs/1602.05450 (2016). arXiv: 1602.05450. url: http://arxiv.org/abs/1602.05450.

[TA07] Kagan Tumer and Adrian K. Agogino. "Distributed agent-based air traffic flow management". In: IFAAMAS, 2007, p. 255. url: https://doi.org/10.1145/1329125.1329434.

References III

[Tan93] Ming Tan. "Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents". In: ed. by Paul E. Utgoff. Morgan Kaufmann, 1993, pp. 330–337. doi: 10.1016/b978-1-55860-307-3.50049-6. url: https://doi.org/10.1016/b978-1-55860-307-3.50049-6.

[TAW02] Kagan Tumer, Adrian K. Agogino, and David H. Wolpert. "Learning sequences of actions in collectives of autonomous agents". In: ACM, 2002, pp. 378–385. url: https://doi.org/10.1145/544741.544832.

[WS06] Ying Wang and Clarence W. de Silva. "Multi-robot Box-pushing: Single-Agent Q-Learning vs. Team Q-Learning". In: IEEE, 2006, pp. 3694–3699. url: https://doi.org/10.1109/IROS.2006.281729.

References IV

[ZB20] Alexander Zai and Brandon Brown. *Deep reinforcement learning in action*. Manning Publications, 2020.