

# Multiagent Reinforcement Learning: Rollout and Policy Iteration (Bertsekas, 2020)

Mikalai Korbit

IMT School for Advanced Studies Lucca

February 20, 2021

# Outline

MARL Overview

Multiagent Rollout

Extensions

Conclusion

# MARL Overview

# Motivation

## Multi-agent systems are ubiquitous

Eg. fleet of drones, factory robots, self-driving cars.

## Recent advances in RL applications

Eg. AlphaGo/AlphaZero, playing Starcraft, robotic control.

## Utilize modern computer architecture and software frameworks

Eg. cloud computing, stacks of graphics cards, TPUs; PyTorch, OpenAI gyms.

## Benefits of modeling a problem as MARL

Scalability, robustness, faster learning through experience sharing, parallel computation.

# Multi-Agent Reinforcement Learning Problem

Inherits Reinforcement Learning characteristics:

- Learning how to map situations into actions
- Trial-and-error search
- Delayed feedback
- Trade-off between exploration and exploitation
- Sequential decision making
- Agent's actions affect the subsequent data it receives

Adds multi-agent features:

- Actions of one agent influence other agents' rewards
- Communication problem
- Fully cooperative, fully info sharing (DP) vs. partial info sharing
- Curse of dimensionality (more severe than in RL)

## “Bertsekas Dictionary”

Aligns optimal control definitions with the RL-world:

- Maximize value  $\rightarrow$  minimize cost
- Agent  $\rightarrow$  Decision maker or controller
- Action  $\rightarrow$  Decision or control
- Environment  $\rightarrow$  Dynamic system
- Learning  $\rightarrow$  Solving a DP-related problem using simulation
- Self-learning (self-play)  $\rightarrow$  Solving a DP problem using simulation-based policy iteration
- Planning vs Learning distinction  $\rightarrow$  Solving a DP problem with model-based vs model-free simulation

# Multi-Agent MDP

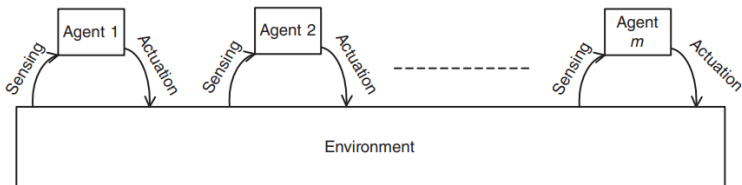


Figure: MARL Problem. Source: Sadhu, Konar (2020)

- All agents see the global state  $s$
- Individual actions:  $u^a \in U$
- State transitions:  $P(s' | s, \mathbf{u}) : S \times \mathbf{U} \times S \rightarrow [0, 1]$
- Shared team reward:  $S \times \mathbf{U} \rightarrow \mathbb{R}$

# Taxonomy

## Cooperative

- The goal of cooperative agents is to achieve a common objective
- Coordination problem

## Competitive

- Zero-sum games (eg. chess, tic-tac-toe)
- Minimax equilibria

## Mixed

- General-sum games (win-win, lose-lose scenarios; eg. pollution model, “what movie to watch?”)
- Nash equilibria

Slide adapted from [2]



# First Attempts to Solve MARL

## Independent Q-Learning (Tan 93)

- Each agent learns its own Q-function
- Each agent treats the other agents as part of the environment
- Independent Q-learning update rule for agent  $a$ :

$$Q(s_t, u_t^a) \leftarrow Q(s_t, u_t^a) + \alpha \left[ r_t + \gamma \max_u Q(s_{t+1}, u) - Q(s_t, u_t^a) \right]$$

## Coordinated Q-Learning (Guestrin et al. 02)

- Stationary learning of joint value function  $Q_{tot}(s, \mathbf{u})$
- Factorize to improve scalability

Slide adapted from [2]

# Multiagent Rollout

# Key Ideas

Deal with the exponential increase in the action space

→ Introduce a form of sequential agent-by-agent one-step lookahead minimization – *multiagent rollout*

Compute the agent actions in parallel

→ Decouple sequential agent-by-agent computation with *precomputed signaling policy* that embodies agent coordination

# The Setting

- $P2_F$  – stochastic discrete-time optimal control problem over a finite horizon, with perfect information on the state
- Fully cooperative
- Tested in Spiders-And-Flies environment

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \dots, N-1$$

$$J_\pi(x_0) = E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}$$

## Policy Iteration and Rollout

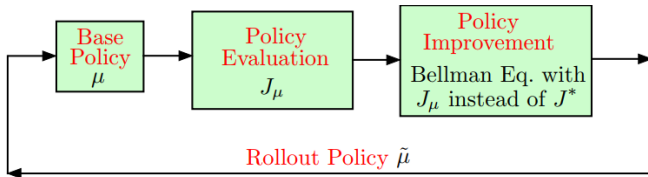


Figure: Policy Iteration Algorithm. Source: Bertsekas (2020)

- Fundamental property: policy improvement

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \quad \forall x_k, k$$

## Standard Rollout Algorithm

- Rollout is one-time policy iteration
- Start with the initial state  $x_0$ , and generate a trajectory:

$$\{x_0, \tilde{u}_0, x_1, \tilde{u}_1, \dots, x_{N-1}, \tilde{u}_{N-1}, x_N\}$$

Where  $\tilde{u}_k$  is

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} E \{g_k(x_k, u_k, w_k) + J_{k+1, \pi}(f_k(x_k, u_k, w_k))\}$$

- Defines rollout policy that possesses **cost improvement property** and **robustness property** (can adapt to changes in data distributions online)
- Works when argmin over small set of  $U$ !

# Standard Rollout for MA Case (All-at-once Rollout)

- The control constraint set becomes the Cartesian product

$$U_k(x_k) = U_k^1(x_k) \times \cdots \times U_k^m(x_k)$$

- Argmin is now computed over  $q^m!$  (where  $q$  is an upper bound to the number of controls in  $U_k$ ,  $m$  is the number of agents)
- Idea: trade-off control space complexity with state space complexity

# One-at-a-time Rollout (Multiagent Rollout)

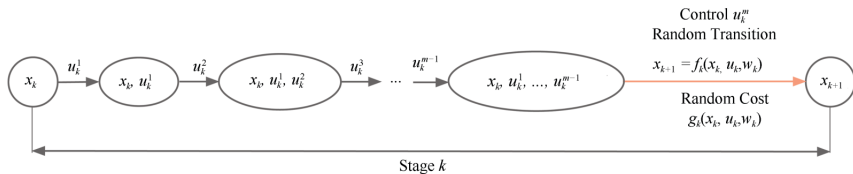


Figure: One-at-a-time action selection. Source: Bertsekas (2020)

1. Break down  $u_k$  into the sequence of  $m$  actions:  $u_k^1, u_k^2, \dots, u_k^m$
2. Introduce artificial states  
 $(x_k, u_k^1), (x_k, u_k^1, u_k^2), \dots, (x_k, u_k^1, \dots, u_k^{m-1})$
3.  $u_k^m$  marks the transition to the new state  $x_{k+1} = f(x_k, u_k, w_k)$  incurring cost  $g_k(x_k, u_k, w_k)$



## Benefits of the Multiagent Rollout Algorithm

Past controls determined by the rollout policy, and the future controls determined by the base policy!

- Reducing the action space by increasing the state space. Reasonable since Q-factor minimization is performed for just one state at each stage.
- We reduce the computation complexity from  $O(q^m)$  to  $O(qm)$ ,  $q = |U|$
- In addition to that, solves coordination. problem.
- Preserves **cost improvement property** (see Bertsekas, part II.D for proof by induction for  $m = 2$ ).

# Multiagent Rollout Assumptions

1. All agents have access to the current state  $x_k$ ;
2. There is an order in which agents compute and apply their local controls;
3. There is “intercommunication” between agents, so that agent  $l$  knows the local controls  $u_k^1, u_k^2, \dots, u_k^{l-1}$  computed by the predecessor agents  $1, 2, \dots, l-1$  in the given order.

## Ordering of Agents

- Instead of predefined or random order, at each step  $k$  optimize over single agent's Q-factors.
- Simulate  $m$  sequences where each agent acts first, select the one with minimal Q-factor, “compete” for the second place with  $m - 1$  agents, etc.
- Total number of minimizations:

$$m + (m - 1) + \cdots + 1 = \frac{m(m + 1)}{2}$$

- Computations can be parallelized.

# Extensions

# Approximate Policy Iteration with Agent-by-Agent Policy Improvement

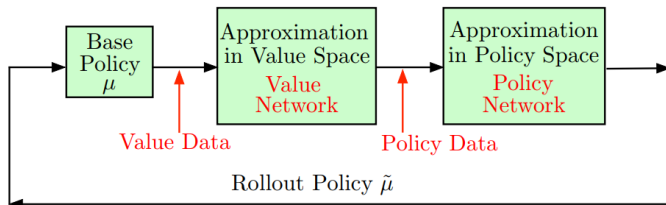


Figure: Approximate Policy Iteration. Source: Bertsekas (2020)

- Approximate policy improvement property: With approximations, policy improvement holds approximately
- If a single policy iteration is done (rollout), no need to train value and policy networks
- Multiple policy iterations can be done only with off-line training

# Parallel Computation of Agents' Controls

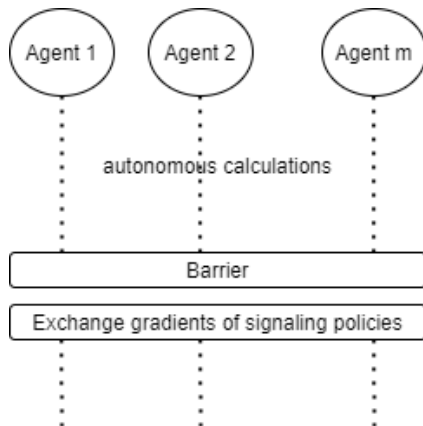
- Reminder: parallel computation vs. action coordination.
- First Attempt: since the agent  $I$  does not know the rollout controls for the agents  $1, \dots, I-1$ , uses the controls  $\mu_k^1(x_k), \dots, \mu_k^{I-1}(x_k)$  of the base policy in their place.
- Drawback: does not preserve cost improvement property.

# Autonomous Multiagent Rollout

- Second Attempt: assume that once the agents know the state, they use precomputed approximations to the control components of the preceding agents, and compute their own control components in parallel and asynchronously – **autonomous multiagent rollout**.
- How to compute approximations? Train a neural network off-line training with training samples generated through the rollout policy – **signaling policy**.
- Use base and signaling policies to generate a rollout policy  $\tilde{\pi} = \{\tilde{\mu}_0, \dots, \tilde{\mu}_{N-1}\}$  autonomously in parallel.

# Synchronized Autonomous Multiagent Rollout

- What if we allow periodic updates of the signaling policies?





# Conclusion

# Conclusion

- MARL problems are especially prone to the curse of the dimensionality problem;
- We could reduce the action space by allowing agent-be-agent updates;
- We could parallelize computations by adding a signaling policy (precalculated offline);
- Multi-agent rollout can be extended with approximate policy iteration.

# References



Dimitri Bertsekas – Multiagent Reinforcement Learning: Rollout and Policy Iteration (2020). Web:  
[https://web.mit.edu/dimitrib/www/Multiagent\\_Sinica\\_2020.pdf](https://web.mit.edu/dimitrib/www/Multiagent_Sinica_2020.pdf)



Shimon Whiteson – Multi-Agent Reinforcement Learning Reinforcement (July 2019) [Eastern European Machine Learning Summer School Seminar].



Arup Kumar Sadhu, Amit Konar – Multi-Agent Coordination, A Reinforcement Learning Approach (2020).

Thanks for  
your attention!