Multi-agent System and Application

Agent Modeling

Xiangfeng Wang

Multi-Agent Artificial Intelligence Laboratory,
School of Computer Science and Technology,
East China Normal University

2021 Spring

Table of Contents

Table of Contents

Introduction

- The core issue of multi-agent reinforcement learning is the development of autonomous agents that can interact effectively with other agents.
- An important aspect of such agents is the ability to reason about the **behaviors, goals, and beliefs** of the other agents.
- This reasoning takes place by constructing *models* of the other agents.

Introduction: The Key Concepts

- **Model.** In general, a model is a function which takes as input some portion of the observed interaction history, and returns a prediction of some property of interest regarding the modeled agent.

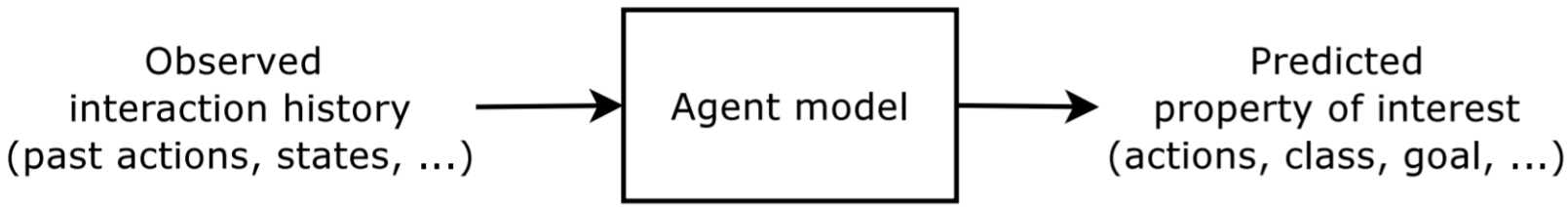

Figure 1: General agent model.

Introduction: The Key Concepts (Cont.)

- **Interaction history.** The interaction history may contain information such as the past actions that the modeled agent took in various situations.
- **Properties of interest.** Properties of interest could be the future actions of the modeled agent, what class of behavior it belongs to (e.g. "defensive", "aggressive"), or its current goals and plans.

Introduction: The Key Concepts (Cont.)

- **Agent Modeling.** The process of constructing models of other agents, sometimes referred to as agent modeling or opponent modeling[1], often involves some form of learning since the model may be based on information observed from the current interaction and possibly data collected in past interactions.

[1] Because much of the early work was developed in the context of competitive games such as Chess, the term "opponent modeling" was established to refer to the process of modeling other agents, and is still used by many researchers.

Introduction: The Usage of Models

- An autonomous agent can utilize such a model in different ways, but arguably the most important one is to *inform* its decision making.
- If the model makes predictions about the **actions** of the modeled agent[2], then the modeling agent can incorporate those predictions in its planning procedure to optimize its interaction with the modeled agent.
- Instead the model can also make predictions about the **class** of behavior of the modeled agent, then the modeling agent could choose a precomputed strategy which it knows to work well against the predicted class.

[2]We will use the term "modeling agent" to refer to the agent which is carrying out the modeling task, and "modeled agent" or "other agent" to refer to the agent which is being modeled.

Introduction: The Usage of Models (Cont.)

- Besides informing decisions, an agent model can also be used for other purposes.
- For example, an intelligent tutoring system could use a model of a specific human player in games such as Chess to identify and point out weaknesses in the human's play.

Introduction: Typical Algorithms

- Many different modeling techniques now exist which vary widely in their underlying assumptions and methodology, largely due to the different needs and constraints of the sub-communities within which they were developed.
- Current methodologies include learning detailed models of an agent's decision making as well as reasoning about spaces of such models; inferring an agent's goals and plans based on hierarchical action descriptions; recursive reasoning to predict an agent's state of mind and its higher-order beliefs about other agents; and many other approaches.

Introduction: Typical Algorithms (Cont.)

We will introduce some typical algorithms for above techniques, including

- Informing decisions.
  - Actions modeling: Deep Policy Inference Q-Network (Hong et al., **AAMAS 2018**)
  - Goals modeling: Modeling Others using Oneself (Raileanu et al., **ICML 2018**)
  - Policies modeling: Learning with Opponent Learning Awareness (Foerster et al., **AAMAS 2018**)
  - Recursive reasoning: Machine Theory of Mind (Rabinowitz et al., **ICML 2018**)
- Stabilizing decisions.
  - Worst cases modeling: Minimax deep deterministic policy gradient (Li et al., **AAAI 2019**)

Table of Contents

Deep Policy Inference Q-Network

- Deep policy inference Q-network (DPIQN) aims at training and controlling a single agent to interact with the other agents in an MAS, using only high-dimensional raw observations (e.g., images).
- DPIQN is built on top of the famous deep Q-network (DQN), and consists of three major parts: a feature extraction module, a Q-value learning module, and an auxiliary policy feature learning module.
- The former two modules are responsible for learning the Q values, while the latter module focuses on learning a hidden representation from the other agents' next actions.

Deep Policy Inference Q-Network

- The learned hidden representation is named as "policy features", and is incorporated into the Q-value learning module to derive better Q values.
- An enhanced version of DPIQN, called deep recurrent policy inference Q-network (DRPIQN), is proposed for handling partial observability resulting from the difficulty to directly deduce or infer the other agents' next actions from only a few observations.

Reinforced & Differentiable Inter-Agent Learning

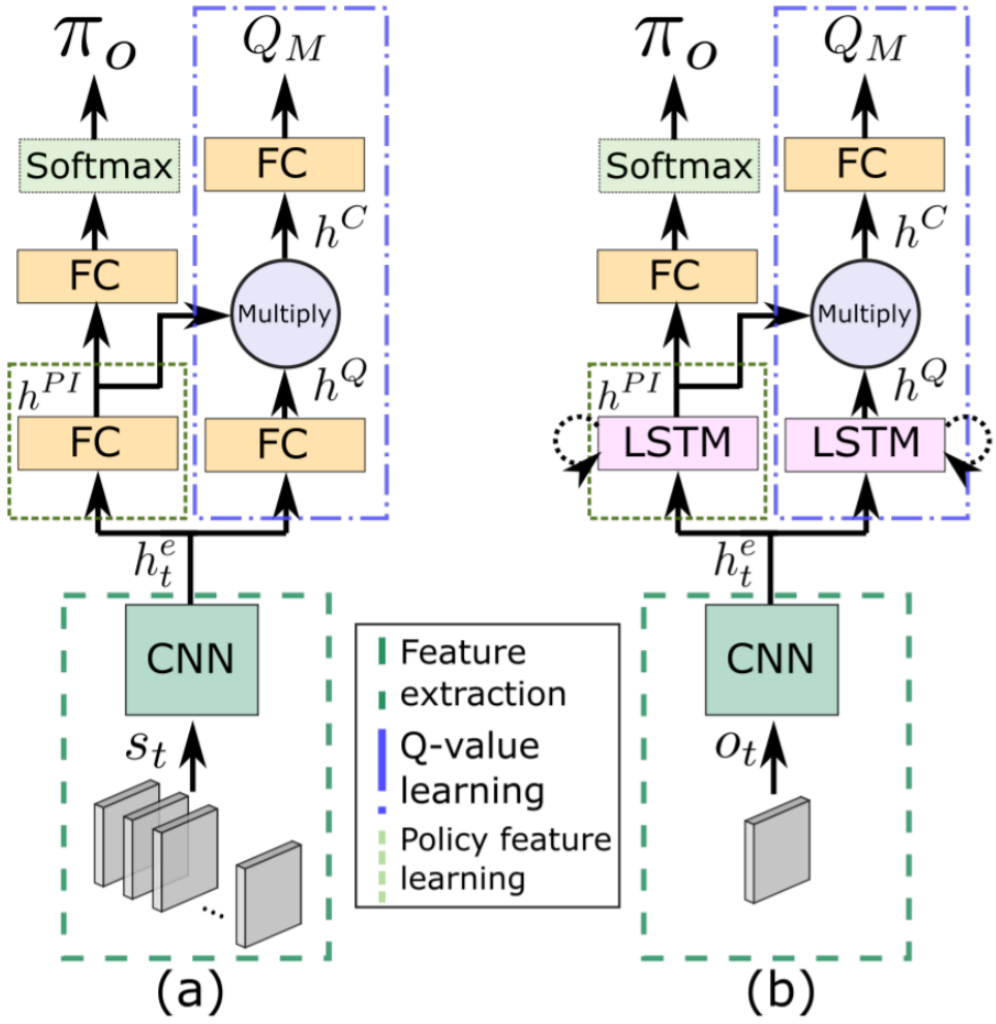

Figure 1: Architectures of DPIQN (a) and DRPIQN (b).

Deep Policy Inference Q-Network

- DPIQN learns a better Q value by constructing an auxiliary task.
- The auxiliary task is a supervised learning task, which uses the historical actions of the other agents to predict the actions they will take at the next moment.
- The above only shows the situation where there are two agents in the environment. When there are multiple agents, we only need to increase the output heads of the network.

Table of Contents

Modeling Others using Oneself

- The problem is formulated as a (not-necessarily zero-sum) two-player stochastic game.
- The agents have full visibility of the environment, but no explicit knowledge about other agents' goals and there is no communication channel.
- The reward received by each agent at the end of an episode depends on the goals of both agents, so the optimal policy of each agent must take into account both of their goals.
- A new approach is proposed for estimating the other agents' unknown goals from their behavior and using those estimates to choose actions.

Modeling Others using Oneself

- The key idea is that as a first approximation, to understand what the other player in the game is doing, an agent should ask itself "**what would be my goal if I had acted as the other player had?**".
- This idea is instantiated by parameterizing the agent's action and value functions with a (multi-layer recurrent) neural network that takes the state and a goal as an input.
- As the agent plays the game, it infers the other agent's unknown goal by directly optimizing over the goal (**using its own action function**) to maximize the likelihood of the other's actions.

Modeling Others using Oneself

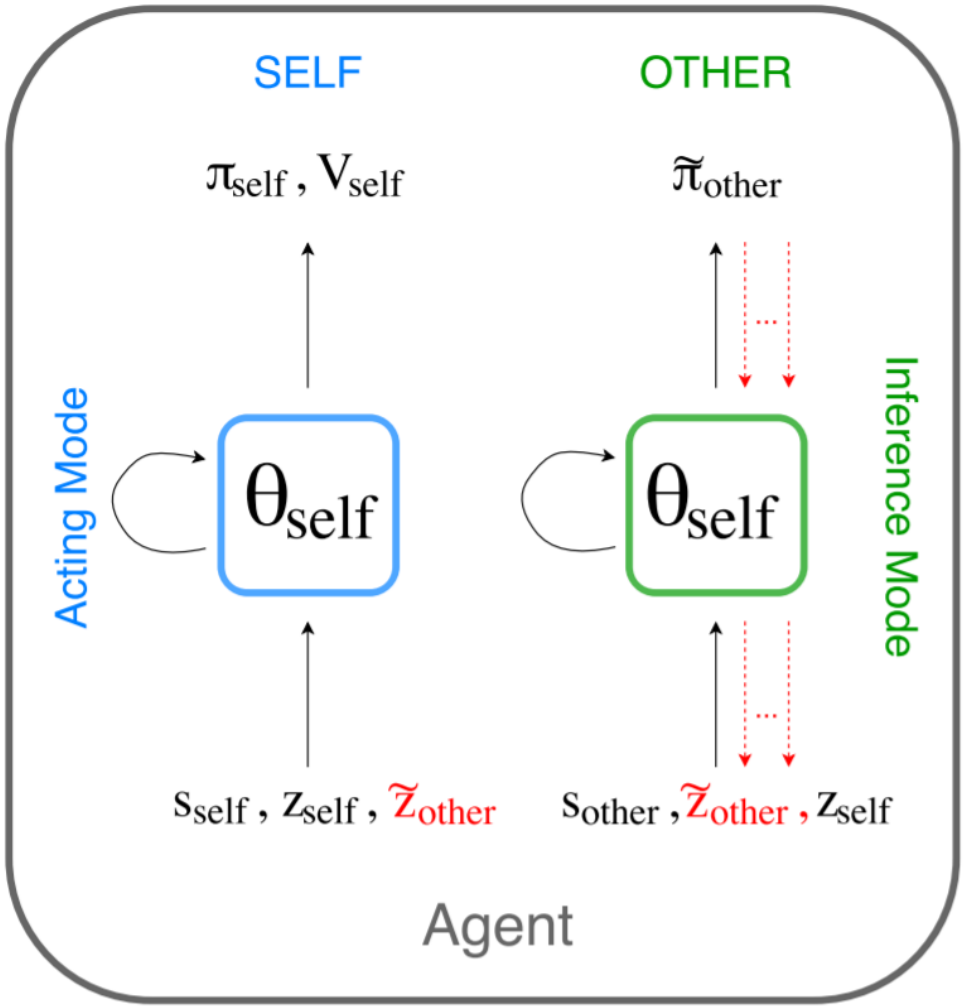

Figure 2: The Self Other-Model (SOM) architecture for a given agent.

Modeling Others using Oneself

- At each step of the game, the agent needs to infer $\tilde{z}_{\text{other}}$ in order to input its estimate into **acting network** and choose its action.
- For this purpose, at each step, the agent observes the other taking an action and, at the next step, the agent uses the previously observed action of the other as supervision, in order to backpropagate through **inference network** and optimize over $\tilde{z}_{\text{other}}$.
- The agents' policies are parametrized by **long short-term memory (LSTM) cells** with two fully-connected linear layers, and exponential linear unit (ELU) activations.

Table of Contents

Learning with Opponent Learning Awareness

- The LOLA learning rule includes an additional term that accounts for the impact of one agent's policy on the learning step of the other agents.

LOLA: Naive Learner

- A Naive Learner (NL) iteratively optimizes for its own expected total discounted return:

$$\theta^1_{i+1} = \text{argmax}_{\theta^1} V^1\left(\theta^1, \theta^2_i\right),$$
$$\theta^2_{i+1} = \text{argmax}_{\theta^2} V^2\left(\theta^1_i, \theta^2\right).$$

- The naive learners apply the gradient ascent update rule as follows:

$$\theta^1_{i+1} = \theta^1_i + f^1_{\text{nl}}\left(\theta^1_i, \theta^2_i\right),$$
$$f^1_{\text{nl}} = \nabla_{\theta^1_i} V^1\left(\theta^1_i, \theta^2_i\right) \cdot \delta,$$

Learning with Opponent Learning Awareness

- A LOLA learner optimizes its return under one step look-ahead of opponent learning: Instead of optimizing the expected return under the current parameters, $V^1\left(\theta_i^1, \theta_i^2\right)$, a LOLA agent optimises $V^1\left(\theta_i^1, \theta_i^2 + \Delta\theta_i^2\right)$, which is the expected return after the opponent updates its policy with one naive learning step, $\Delta\theta_i^2$.
- A first-order Taylor expansion results in:

$$V^1\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2 + \Delta\boldsymbol{\theta}^2\right) \approx V^1\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2\right) + \left(\Delta\boldsymbol{\theta}^2\right)^T \nabla_{\boldsymbol{\theta}^2} V^1\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2\right).$$

- LOLA learners attempt to actively influence the opponent's future policy update, and explicitly differentiate through the $\Delta\theta^2$ with respect to θ^1.

Learning with Opponent Learning Awareness

- By substituting the opponent's naive learning step, we obtain our LOLA learning rule which includes a second order correction term:

$$f^1_{\text{lola}}\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2\right) = \nabla_{\boldsymbol{\theta}^1} V^1\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2\right) \cdot \delta + \left(\nabla_{\boldsymbol{\theta}^2} V^1\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2\right)\right)^T \nabla_{\boldsymbol{\theta}^1} \nabla_{\boldsymbol{\theta}^2} V^2\left(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2\right) \cdot \delta\eta.$$

Table of Contents

Machine Theory of Mind

- Theory of mind is part of a group of recursive reasoning approaches in which agents have explicit beliefs about the mental states of other agents.
- The mental states of other agents may, in turn, also contain beliefs and mental states of other agents, leading to a nesting of beliefs.
- Machine Theory of Mind (ToMnet) starts with a simple premise: *when encountering a novel opponent, the agent should already have a strong and rich prior about how the opponent should behave*.

Machine Theory of Mind

ToMnet has an architecture composed of three networks:

- A character network that learns from historical information.
- A mental state network that takes the character output and the recent trajectory.
- The prediction network that takes the current state as well as the outputs of the other networks as its input.

The output of the architecture is open for different problems but in general its goal is to predict the opponent's next action.

Machine Theory of Mind

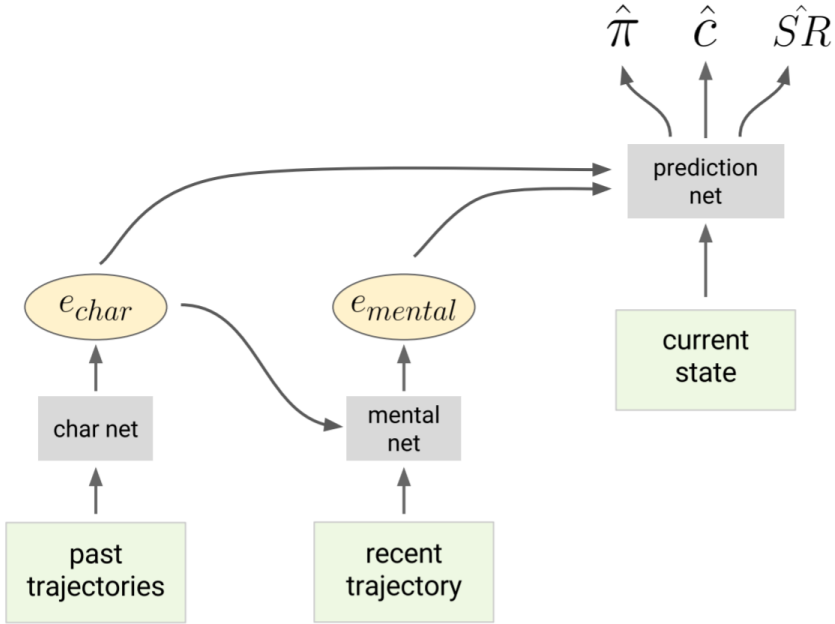

Figure 3: ToMnet architecture. The character net parses an agent's past trajectories from a set of POMDPs to form a character embedding, e_{char}. The mental state net parses the agent's trajectory on the current episode, to form an embedding of its mental state, e_{mental} . These embeddings are fed into the prediction net, which is then queried with a current state. This outputs predictions about future behavior, such as next-step action probabilities ($\hat{\pi}$), probabilities of whether certain objects will be consumed ($\hat{c}$), and predicted successor representations ($\widehat{SR}$).

Table of Contents

Minimax Deep Deterministic Policy Gradient

- Minimax is a paramount concept in game theory that is roughly described as minimizing the worst case scenario (maximum loss).
- This method take the minimax idea as an approach to robustify learning in multi-agent environments so that the learned robust policy should be able to behave well even with strategies not seen during training.
- MADDPG algorithm is extended to Minimax Multi-agent Deep Deterministic Policy Gradients (M3DDPG), which updates policies considering a worst-case scenario: assuming that all other agents act adversarially.

Minimax Deep Deterministic Policy Gradient

- This yields a minimax learning objective which is computationally intractable to directly optimize.

$$
\begin{aligned}
J_{\mathrm{M}}(\theta_i) &= \mathbb{E}_{s\sim\rho^{\mu}}\left[R_i\right] \\
&= \min_{a_{j\neq i}^{t}} \mathbb{E}_{s\sim\rho^{\mu}}\left[\sum_{t=0}^{T}\gamma^t r_i\left(s^t, a_1^t, \ldots, a_N^t\right)\Big|_{a_i^t=\boldsymbol{\mu}\left(o_i^t\right)}\right] \\
&= \mathbb{E}_{s^0\sim\rho}\left[\min_{a_{j\neq i}^{0}} Q_{\mathrm{M},i}^{\boldsymbol{\mu}}\left(s^0, a_1^0, \ldots, a_N^0\right)\Big|_{a_i^0=\boldsymbol{\mu}\left(o_i^0\right)}\right].
\end{aligned}
$$

Minimax Deep Deterministic Policy Gradient

- M3DDPG addresses this issue by taking ideas from robust reinforcement learning which implicitly adopts the minimax idea by using the worst noise concept.
- The main ideas of MAAL can be summarized in two steps:
  - Approximate the non-linear Q function by a locally linear function;
  - Replace the inner-loop minimization with a 1-step gradient descent.

Minimax Deep Deterministic Policy Gradient

- Eventually the following formulation is derived:

$$\nabla_{\theta_i} J(\theta_i) =$$
$$\mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}} \left[\begin{array}{c} \nabla_{\theta_i} \boldsymbol{\mu}_i (o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{\mathrm{M},i} (\mathbf{x}, a^\star_1, \ldots, a_i, \ldots a^\star_N) \mid \\ a_i = \boldsymbol{\mu}_i (o_i) \\ a^\star_j = a_j + \hat{\epsilon}_j, \quad \forall j \neq i \\ \hat{\epsilon}_j = -\alpha_j \nabla_{a_j} Q^{\boldsymbol{\mu}}_{\mathrm{M},i} (\mathbf{x}, a_1, \ldots, a_N) \end{array} \right].$$