Symbolic and Neural Approaches for Learning Other Agents' Intentional Models using Interactive POMDPs

by

Yanlin Han
B.A. (Shandong University of Science and Technology) 2009
M.E. (University of Science and Technology of China) 2011
M.S. (Clemson University) 2012

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2018

Chicago, Illinois

Defense Committee:
Piotr Gmytrasiewicz, Chair and Advisor
Bing Liu
Brian Ziebart
Xinhua Zhang
Erdem Koyuncu, Electrical and Computer Engineering

To my family, for their unconditional love and support.

ACKNOWLEDGMENTS

I would like to thank my thesis committee members for their support and assistance in all areas that helped me achieve my research goals. I have learned a great deal from them in classes, researches, and discussions, and I deeply cherish the time I have spent working with them. Special thanks to my academic advisor Piotr Gmytrasiewicz, for his guidance during my PhD study. I am grateful for his efforts to get me to work on different research topics and reach the depth . He has also been extremely helpful in formulating my written work in a more comprehensive way.

I would also like to thank my colleagues and friends who have been helping me throughout the years. From laboratory discussions to leisure time activities, they helped me build the academic path that culminates in this dissertation. Their companionship will be one of the best memories during my study and life in UIC.

Most importantly, I wish to thank my wife, my daughter and my parents. Without their unconditional support, I could not accomplish any single goal I set in the PhD program.

YH

CONTRIBUTION OF AUTHORS

Chapter 4 and 5 extend from a published manuscript (Han and Gmytrasiewicz, 2017) for which I was the primary author. My advisor Professor Piotr Gmytrasiewicz contributed to discussions about the ideas of sampling algorithms and assisted in revising the manuscript.

Chapter 6 extends from a manuscript under review (Han and Gmytrasiewicz, 2018) for which I was the primary author. My advisor Professor Piotr Gmytrasiewicz contributed to discussions about the various approaches for the implementation.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
I-PF	Interactive Particle Filter
I-POMDP	Interactive Partially Observable Markov Decision Process
IPOMDP-Net	Interactive Partially Observable Markov Decision Process Network
IS	Interactive State
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
PF	Particle Filter
POMDP	Partially Observable Markov Decision Process
QMDP	Q-values of the underlying Markov Decision Process (of a POMDP)
QMDP-Net	QMDP Network
RNN	Recurrent Neural Network
SE	State Estimation function

SUMMARY

Interactive partially observable Markov decision processes (I-POMDPs) provide a principled framework for planning and acting in a multi-agent, partially observable and stochastic environment. It incorporates models of other agents in the state space and extends POMDPs to multi-agent settings. I-POMDPs augment POMDP belief spaces with nested hierarchical belief structures which represent an agent's belief about the environment and other agents, and about their beliefs about the environment and others' beliefs. In order to plan optimally in such settings using I-POMDPs, we propose an approach that effectively uses Bayesian inference and sequential Monte Carlo sampling to learn others agents' intentional models which ascribe to them beliefs, preferences and rationality in action selection.

In this symbolic approach, agents maintain beliefs over intentional models of other agents and make sequential Bayesian updates using observations. The modeling agent does not have any information about other agents' beliefs, and transition, observation and reward functions, it relies on learning indirectly from observations about the environment. To deal with the complexity of the hierarchical belief space, we have devised a customized interactive particle filter (I-PF) to descend the belief hierarchy, parametrize others' models, and sample all model parameters at each nesting level.

We have also devised a neural network approximation of the I-POMDP framework, in which the belief update, value function, and policy function are implemented by various deep neural networks (DNNs). The I-POMDP network (IPOMDP-net) embeds the I-POMDP model and the QMDP planner in its architecture, and can be trained end-to-end in a reinforcement learning fashion. The performance of our model-based IPOMDP-net is compared with the other state-of-the-art model-free network.

SUMMARY (Continued)

Empirical results show that our Bayesian learning approach accurately learns models of other agents and performs superiorly compared with methods that use subintentional models. It serves as a generalized Bayesian learning algorithm that learns other agents' beliefs, strategy levels, and transition, observation and reward functions. It effectively mitigates the belief space complexity of I-POMDPs. The IPOMDP-net provides an effective and efficient end-to-end neural computing architecture for multi-agent planning using I-POMDPs. Moreover, we show that, in a reinforcement learning setting, the model-based IPOMDP-net which learns to plan outperforms the model-free network which learns reactive policies.

CHAPTER 1

INTRODUCTION

Partially observable Markov decision processes (POMDPs) (1) is a general decision-theoretic framework for planning and acting under uncertainty in a partially observable, stochastic environment. To act rationally in such an environment, an agent constantly maintains beliefs over the state of the environment and sequentially select the optimal actions that maximize the expected value of future rewards. Thus, solutions of POMDPs map an agent's beliefs to actions. In order to apply POMDPs to multi-agent settings, the impacts of other agents' actions are usually treated as noise and embedded into the state transition function. Examples of such POMDPs are Utile Suffix Memory (2), infinite generalized policy representation (3), and infinite POMDPs (4). In such approaches, an agent's beliefs of other agents are not contained in the POMDP solutions.

Interactive POMDPs (I-POMDPs) (5) generalize POMDPs to multi-agent settings by augmenting the plain beliefs in POMDP with models of other agents. As a result, the I-POMDP replaces the POMDP belief space with a hierarchical interactive belief structure. This extended hierarchical belief structure represents a modeling agent's belief about the physical environment and other agents, in which it represents the modeled agents' beliefs about the environment and others' beliefs, and so on. Such interactive belief hierarchy can be nested infinitely, but the finitely nested counterparts are usually used in practice due to their computability. There are two types of models incorporated in the I-POMDP belief space: intentional models and subintentional models. An intentional model ascribes beliefs, preferences, and rationality to modeled agents (5), while a subintentional model, such as finite state controllers (6), does

not ascribe such components to other agents. Thus, solutions of I-POMDPs are mapping from an agent's belief space about the environment and other agents' models to actions. It has been shown (5) that the extension of POMDPs to I-POMDPs results in a dominant value function over one obtained from implicitly embedding others' impacts as noise, the added sophistication of modeling others as rational agents implies the superior modeling capability of I-POMDPs in a multi-agent environment.

However, the augmentation of I-POMDPs' interactive beliefs results in a drastic increase of the belief space complexity. Since the number of agents' models grows exponentially as the nesting level increases, the complexity of the belief representation is proportional to belief state dimensions, which is known as the curse of dimensionality. For POMDPs, it has been proved that exact solutions are PSPACE-complete for finite time horizon and undecidable for infinite time horizon (7). For generalized I-POMDPs, an I-POMDP model may contain multiple POMDP or I-POMDP models of other agents, depending on the actual nesting level. Therefore, the time complexity of I-POMDPs is at least PSPACE-complete for finite time horizon and undecidable for infinite time horizon. Due to these complexities, a solution which accounts for an agent's belief over an entire intentional model has not been implemented up to date. There are partial solutions that depend on what is known about other agents' beliefs about the physical states (8), but they do not include the state of an agent's knowledge about others' strategy level, and reward, transition, and observation functions. Indirect approaches such as subintentional finite state controllers (6) do not include any of these elements either. To unleash the full modeling power of intentional models and mitigate the aforementioned complexities, a robust approximation algorithm is needed. The purpose of this algorithm is to compute the nested interactive belief over elements of

the intentional models and predict other agents' actions. It is crucial to the trade-off between solution quality and computational complexity for solving I-POMDPs.

To address these issues, we propose a Bayesian learning method that utilizes customized sequential Monte Carlo sampling (9) to obtain approximate solutions to I-POMDPs. We assume that the modeling agent maintains beliefs over intentional models of other agents and make sequential Bayesian updates using observations from the environment. While in multi-agent settings, others agents' models other than their beliefs are usually assumed to be known, in our approach the modeling agent may start off without information about others' beliefs, strategy levels, and transition, observation, and reward functions. It only relies on learning indirectly from observations about the environment, which is influenced by others agents' actions. Since this Bayesian inference task is analytically intractable due to the requirement of computing high dimensional integrations, we have devised a customized sequential Monte Carlo method, extending the interactive particle filter (I-PF) (8) to the entire intentional model space. The main idea of this method is to descend the nested belief hierarchy, parametrize other agents' model functions, and sample all model parameters at each nesting level according to observations.

On the other hand, there have been recent advances which apply deep reinforcement learning methods in sequential decision making problems. One method is to learn policies directly, such as using model-free reinforcement learning methods, in which the mapping from (belief) state space to optimal action space is learned. While the model-free policy learning can be end-to-end, it lacks the model information for effective generalization. Particularly, in the multi-agent domain, other agents' actions impact the environment, which indirectly impact our policies. Therefore, we prefer a model-based method,

since having a model of other agents helps our reasoning about their actions and consequently benefits our decision making.

Based on the symbolic Bayesian learning approach discussed above, we propose a neural network architecture, the IPOMDP-net, for multi-agent planning under partial observability using I-POMDPs. We extends the QMDP-net (10) architecture to a multi-agent domain by combining an I-POMDP model with a QMDP (11) planning algorithm and embedding both in a recurrent neural network. We implement the IPOMDP-net on GPU-based computing devices and comparing its performance with the corresponding symbolic I-POMDP.

Moreover, we apply the same network architecture in reinforcement learning problems of various sizes and dimensions. We train our model-based network, starting from randomly initialized weights, in a reinforcement learning fashion assuming the reward is obtainable. We then evaluate the trained IPOMDP-net by comparing it with another model-free neural network. We show empirical results that our model-based network outperforms the state-of-the-art model-free network in different tasks due to its capability of learning to plan using embedded models.

Overall, the symbolic Bayesian learning approach successfully recovers others agents' models over the intentional model space which contains their beliefs, strategy levels, and transition, observation, reward functions. It extends existing I-POMDP belief update to larger model space, serving as a generalized Bayesian learning method for multi-agent systems in which other agents' beliefs, strategy levels, and transition, observation and reward functions are unknown. We mitigate the belief space complexity of I-POMDPs by approximating the exact Bayesian inference using a customized sequential Monte Carlo sampling method. On the other hand, the neural approach provides a general neural computing

architecture for multi-agent planning using I-POMDPs. Besides the capability for planning problems in which the rewards are unobserved, the IPOMDP-net is fully differentiable and allows for end-to-end training, therefore, the same architecture can be used in reinforcement learning problems in which the rewards are observed. In such settings, the IPOMDP-net trained on small-size problems generalizes more effectively to larger difficult settings compared with model-free networks. It suggests that, in a multi-agent reinforcement learning setting, having a model of other agents benefits our decision-making, resulting in a policy of higher quality and better generalizability.

1.1 Summary of Contributions and Results

The major goals of this thesis research are to develop an generalized Bayesian learning method to learn other agents' intentional models using I-POMDPs and devise an neural approximation of I-POMDP and apply its network architecture in deep reinforcement learning problems. These goals are achieved by means of the following contributions:

- We design and implement an model-based agent which is capable of learning others' intentional models using the I-POMDP framework. The agent explicitly models other agents' beliefs, preferences and rationality in action selection, providing a generalized Bayesian learning framework in a partially observable, stochastic and multi-agent environment.
- Due to the complexity of the hierarchical belief structure and the high-dimensional belief space, exact computations of belief updates are intractable. We have devised a customized Monte Carlo sampling algorithms extended from the I-PF in order to make the I-POMDP framework computationally feasible.

- With the help of aforementioned sampling algorithms, we are able to experiment with deeper strategy (nesting) levels and show the performance increase. We give analysis of the learning quality as well as the computational cost, providing a better understanding of the relation between depth and improvement of the intentional modeling.
- To make an interesting comparison, we also develop a neural approach that implements the I-POMDP framework as a neural network which can be computed on specialized neural computing hardware such as GPUs. We compare the I-POMDP network (IPOMDP-net) with its symbolic counterpart on same problems used in the same settings where the models are given and rewards are unobserved.
- We have also devised a reinforcement learning algorithm to train the same network architecture as in the IPOMDP-net with randomly initialized weights and observable rewards assumption. We demonstrate the results using applications on classic games from the POMDP community, such as the tiger problem (1), the UAV problem (8), and the maze problem (13) and its variations. Empirical results show that our model-based network generalize better than the state-of-the-art model-free network.

1.2 Structure of the Thesis

The remaining sections of this thesis are organized as follows:

- In Chapter 2, we briefly review the related work in multi-agent POMDPs and deep reinforcement learning.

- In Chapter 3, we provide the background knowledge of POMDPs, I-POMDPs, Monte Carlo sampling algorithms, and deep reinforcement learning methods.
- In Chapter 4, we introduce in details the customized sampling algorithms devised for interactive belief updates on the intentional model space and illustrate them using the multi-agent tiger problem as an example.
- In Chapter 5, we give the experimental details, briefly demonstrate the learning process, and analyze the solution quality, convergence, and computational cost with further experiments on deeper strategy levels.
- In Chapter 6, we introduce the neural I-POMDP implementation, the IPOMDP-net, in which major components are translated into different neural network architectures. We test its planning performance with its symbolic counterpart in multi-agent tiger and UAV problems, with known models and unobservable reward signals.
- In Chapter 7, we discuss our work of applying the same network architecture as in the IPOMDP-net to reinforcement learning problems where models are assumed unknown and reward signals are observable. We show the training algorithm, experimental results, and a simple visualization.
- Chapter 8 concludes the thesis with a brief summary and some future research directions.

CHAPTER 2

RELATED WORK

The partially observable Markov decision process (POMDP) is a general framework to model a variety of real-world sequential decision processes. To act rationally in a single-agent, partially observable and stochastic environment, the agent keeps track of the physical state of the environment by maintaining a probability distribution of the state at each time step. With the ever-updating belief distribution, the agent sequentially selects the optimal actions that maximize the expected value of future rewards. POMDPs can be directly applied into multi-agent settings in a straightforward manner, however, doing so treats impacts of others' actions as noise and folds them into the state transition. Examples of such direct applications are Bayes-adaptive POMDPs (14), infinite generalized policy representation (3), and infinite POMDPs (4).

2.1 Bayesian Learning of Other Agents' Models

Within the last decade, research on multi-agent sequential decision-making has made substantial progresses. Many frameworks in this domain extend the single-agent POMDP model to multi-agent settings and provide theoretical and practical feasibility. Among such frameworks, a large body of work has been on decentralized POMDPs (DEC-POMDPs) (15), which generalize POMDPs to multiple decentralized agents (16). DEC-POMDPs are generally viewed as an "objective" approach that is suitable for modeling agents in a fully cooperative setting, as Dec-POMDP assumes common rewards

among agents. In another "subjective" approach, the interactive POMDPs (I-POMDPs) (5), each agent maintains beliefs about both the physical states of the world and models of other agents.

I-POMDPs incorporate models of other agents in the belief state space, forming an augmented belief structure which represents an agents' belief about the environment as well as other agents, which in turn contains the modeled agents' beliefs about the environment and others, and so on. Therefore, I-POMDPs generalize POMDPs to multi-agent settings by replacing the plain beliefs in POMDP with the resultant hierarchical belief structure called the interactive belief. There are two types of models incorporated in the new interactive belief space: intentional models and subintentional models. An intentional model ascribes to other agents beliefs, preferences, and rationality in action selection (5); a subintentional model, such as finite state controllers (6), does not include such sophistication. Solutions of I-POMDPs are therefore mappings from an agent's belief space, which is about the physical state as well as other agents, to action space.

Recently there has been many advances on modeling and learning other agents' models in multi-agent systems, but none of them have managed to learn over the entire space of others' models in the formulation of multi-agent POMDPs. In particular, the Bayes Adaptive I-POMDPs (BA-IPOMDPs) generalizes the Bayes Adaptive POMDPs(BA-POMDPs) (14) to multi-agent domain by incorporating model learning in I-POMDPs through Bayesian reinforcement learning (17) . BA-IPOMDPs impose additional hyper parameters of Dirichlet distributions over actual model parameters to learn, such as those in the transition and observation functions, in order to make Bayesian inference on them. The Bayesian Policy Reuse (BPR) (18) have been used to learn opponents' policies in MDP settings. For

instance, in (19), they have mixed online and offline learning methods in MDP settings. Obviously, the BPR does not deal with partial observability.

Due to the sophistication of the extended belief structure, obtaining optimal solutions in I-POMDP remains difficult. An alternative approach is to use subintentional models that do not explicitly consider others agents' preferences and beliefs. Specifically, in (6), probabilistic deterministic finite-state controllers (PDFCs) are used to approximate others' models. Within the intentional modeling approach, the interactive particle filter (I-PF) (8) generalizes particle filters to the multi-agent setting and maintains a belief over others' beliefs about the physical state of the world. The I-PF assumes that, in other agents' intentional models, all other preferences except beliefs about the physical state are known, which significantly reduces the intentional model space of I-POMDPs to a smaller one. Our customized I-PF extends the I-PF to the entire intentional model space, providing a computational tool for modeling and learning other agents using intentional models on higher strategy levels, which yields better solution quality while still mitigates the belief space complexity.

2.2 Neural Approaches for Sequential Decision Making

On the other hand, the unprecedented success of deep neural networks (DNNs) in supervised learning (20; 21; 22) provides new approaches to decision making under uncertainty. Approximate algorithms utilizing deep neural networks have made dramatic progress on solving large-scale sequential decision making problems. There are two major approaches to learn POMDP policies: the first is to learn models (23; 24) and solve the learned models through planning, the other is to directly learn policies (25; 26). The model learning approach is usually not end-to-end. Although the policy learning

approach can be end-to-end, it does not make use of model information therefore lacks the ability to effectively generalize.

Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been applied to tasks like Atari games (27), robotics (28), and 2D path planning (10). In these tasks, a DNN is trained to approximate a policy function that maps an agent's observations to optimal actions.

The deep Q-network (DQN), which consists convolutional layers for feature extraction and a fully connected layer for mapping features to a probability distribution over actions, tackles many Atari games with the same network architecture (27). DQN is inherently reactive and lacks explicit planning computation (29). The deep recurrent Q-network (DRQN) (31) extends DQN to the partially observable domain by replacing the fully connected layer with a recurrent long short-term memory (LSTM) (30) layer to integrate temporal information.

Furthermore, the idea of embedding specific computational structures in the neural network architecture has gained attention recently. The value iteration network (VIN) started to embed specific computation structures (29), particularly the value iteration algorithm, in the network architecture. The VIN solves MDPs in the fully observable domain, but does not address the issue of partial observability. The Predictron was introduced for value estimation in Markov reward processes (32), and the end-to-end trainable Bayesian filters (33) (34) are developed for probabilistic state estimation. They are essentially neural filtering algorithms and do not deal with decision making. Both (35) and (36) discussed planning under partial observability. The former proposes a planning network of which architecture consists of a hierarchical extension of VIN and therefore only deals with partial observability. The latter learns a model which is trained in a fixed environment, therefore it does not generalize to new environments. The

QMDP-net embeds a POMDP model and a QMDP planning algorithm in a recurrent policy network. The QMDP-net effectively combines model-free learning with model-based planning, and is trained end-to-end through imitation learning(10).

However, all the discussed neural networks are either model-free or for single-agent settings. Unless we model other agents' impact as noise and embed it into the world dynamic, which often leads to inferior solution quality, it is not feasible to directly apply these networks into a multi-agent partially observable domain.

CHAPTER 3

BACKGROUND

In this chapter, we will give a brief overview of related background, which prepares the readers for a better understanding of the algorithms and architectures discussed in the following chapters. We firstly introduce the general frameworks for single and multi agent partially observable Markov decision processes; then we show how the agent's belief can be updated using different sampling algorithms; lastly we talk about the prevalent neural approaches for solving such tasks.

3.1 Partially Observable Markov Decision Processes

A partially observable Markov decision process (POMDP) (1) is a general decision-theoretic framework for optimal planning in a single-agent, partially observable and stochastic domain. A POMDP of agent i is defined as

$$POMDP_i = \langle S, A_i, \Omega_i, T_i, O_i, R_i \rangle \tag{3.1}$$

where

- S is a set of possible environment states,
- A_i is a set of agent i's possible actions,
- Ω_i is a set of agent i's possible observations,
- $T_i : S \times A_i \times S \to [0, 1]$ is agent i's state transition function,
- $O_i : S \times A_i \times \Omega_i \to [0, 1]$ is agent i's observation function,

- $R_i : S \times A_i \rightarrow \mathbb{R}$ is agent i's reward function.

With these definitions, we can represent an agent's belief about the environment as a probability distribution over the physical state S. This belief distribution can be updated by the following equation:

$$b_i^t(s^t) = \alpha O_i(s^t, a_i^{t-1}, o_i^t) \sum_{s^{t-1} \in S} T_i(s^{t-1}, a_i^{t-1}, s^t) b_i^{t-1}(s^{t-1}) \tag{3.2}$$

where α is the normalizing constant. Conveniently, we can denote Equation 3.2 as $b_i^t = SE(b_i^{t-1}, a_i^{t-1}, o_i^t)$ (SE means state estimation) (1).

We can associate the value with a belief state b_i to quantify the value of a belief state, which is composed of the best immediate reward and the discounted expected sum of values of the subsequent belief states:

$$V(b_i) = \max_{a_i \in A_i} \Big\{ \sum_{s \in S} b_i(s) R(s, a_i) + \gamma \sum_{o_i \in \Omega_i} P(o_i | a_i, b_i) V(SE(b_i, a_i, o_i)) \Big\} \tag{3.3}$$

where γ is the discount factor. Or sometimes, the values are associated with actions as well:

$$V(b_i) = \max_{a_i \in A_i} \{Q(b_i, a_i)\} \tag{3.4}$$

The value iteration algorithm uses the Equation Equation 3.3 iteratively to obtain values of belief states for longer time horizons. Then the optimal action, a^*, belongs to the set of optimal actions, $OPT(b_i)$, defined as:

$$OPT(b_i) = \arg\max_{a_i \in A_i} \Big\{ \sum_{s \in S} b_i(s) R(s, a_i) + \gamma \sum_{o_i \in \Omega_i} P(o_i | a_i, b_i) V(SE(b_i, a_i, o_i)) \Big\} \tag{3.5}$$

3.2 Interactive POMDPs

I-POMDPs (5) include models of other agents in the belief space, generalizing POMDPs to multi-agent settings. The resultant hierarchical belief structure represents the modeling agent's belief about the environment as well as other agents, which in turn represents modeled agents' beliefs about the environment and others' beliefs. This hierarchical structure can be nested infinitely in the recursive manner. Here, we focus on the computable counterparts of infinitely nested I-POMDPs: finitely nested I-POMDPs. For simplicity of presentation, we consider two interacting agents i and j. This formalism generalizes to more number of agents in a straightforward manner.

Equation 3.6 defines a finitely nested interactive POMDP of agent i , I-POMDP$_{i,l}$:

$$I\text{-}POMDP_{i,l} = \langle IS_{i,l}, A, \Omega_i, T_i, O_i, R_i \rangle \tag{3.6}$$

where $IS_{i,l}$ is a set of interactive states, defined as $IS_{i,l} = S \times M_{j,l-1}, l \geq 1$. S is the set of physical states of the environment, $M_{j,l-1}$ is the set of possible models of agent j's, and l is the *nesting (strategy) level*. The set of models, $M_{j,l-1}$, can be divided into two classes, the intentional models, $IM_{j,l-1}$, and subintentional models, $SM_{j,l-1}$. Thus, $M_{j,l-1} = IM_{j,l-1} \cup SM_{j,l-1}$.

The *intentional* models, $IM_{j,l-1}$, ascribe beliefs, preferences, and rationality in action selection to other agents, thus they are analogous to *types*, θ_j, used in Bayesian games (37). Agent j's intentional models at level $l-1$, $\Theta_{j,l-1}$, can be defined as $\theta_{j,l-1} = \langle b_{j,l-1}, A, \Omega_j, T_j, O_j, R_j, OC_j \rangle$, where $b_{j,l-1}$ is j's $(l-1)$th level belief, $b_{j,l-1} \in \Delta(IS_{j,l-1})$, and OC_j is j's optimality criterion. An intentional model of agent j can be rewritten as $\theta_{j,l-1} = \langle b_{j,l-1}, \hat{\theta}_j \rangle$, where $\hat{\theta}_j$ includes all but the belief of the intentional model and is called j's *frame*.

The *subintentional* models, $SM_{j,l-1}$, constitute the remaining models in $M_{j,l-1}$. They do not use the notion of rationality to model other agents. Examples of subintentional models are finite state controllers (6), no-information models (38), and fictitious play models (39).

Given the model definitions, the $IS_{i,l}$ can be defined in an inductive manner:

$$\begin{aligned}
&IS_{i,0} = S, && \theta_{j,0} = \{\langle b_{j,0}, \hat{\theta}_j \rangle : b_{j,0} \in \Delta(S)\} \\
&IS_{i,1} = S \times \theta_{j,0}, && \theta_{j,1} = \{\langle b_{j,1}, \hat{\theta}_j \rangle : b_{j,1} \in \Delta(IS_{j,1})\} \\
&\ldots\ldots && \\
&IS_{i,l} = S \times \theta_{j,l-1}, && \theta_{j,l} = \{\langle b_{j,l}, \hat{\theta}_j \rangle : b_{j,l} \in \Delta(IS_{j,l})\}
\end{aligned} \tag{3.7}$$

All remaining components in an I-POMDP are similar to those in a POMDP:

- $A = A_i \times A_j$ is the set of joint actions of both agents,
- Ω_i is the set of possible observations of agent i,
- $T_i : S \times A \times S \to [0, 1]$ is agent i's transition function,
- $O_i : S \times A \times \Omega_i \to [0, 1]$ is agent i's observation function,

- $R_i : IS_i \times A \rightarrow \mathbb{R}$ is agent i's reward function.

Compared with POMDP, the action set in I-POMDP contains joint actions of all agents. Consequently, the observation set may contain indications of other agents' actions, and the new transition, observation and action functions all involve other agents' actions as well.

3.2.1 Interactive Belief Update

Given the definitions above, the interactive belief update can be performed by considering others' actions and anticipated observations:

$$
\begin{aligned}
& b_{i,l}^t(is^t) = Pr(is^t | b_{i,l}^{t-1}, a_i^{t-1}, o_i^t) \\
& = \alpha \sum_{is^{t-1}} b_{i,l}(is^{t-1}) \sum_{a_j^{t-1}} Pr(a_j^{t-1} | \theta_{j,l-1}^{t-1}) T(s^{t-1}, a^{t-1}, s^t) O_i(s^t, a^{t-1}, o_i^t) \\
& \times \sum_{o_j^t} O_j(s^t, a^{t-1}, o_j^t) \tau(b_{j,l-1}^{t-1}, a_j^{t-1}, o_j^t, b_{j,l-1}^t)
\end{aligned} \tag{3.8}
$$

Conveniently, we can denote the belief update function as $b_{i,l}^t = SE_{\theta_i}(b_{i,l}^{t-1}, a_i^{t-1}, o_i^t)$.

Compared with POMDPs, the interactive belief update in I-POMDPs takes into account two additional computations. First, we need to compute the probability of the other agent's actions conditioned on its models, because the physical state now depends on both agents' actions (i.e. the second summation). Second, we need to update the other agent's beliefs based on their anticipated observations (i.e. the third summation).

3.2.2 Value Iteration for I-POMDPs

Similarly to POMDPs, the value associated with an interactive belief state in I-POMDPs can be updated using value iterations:

$$V(\theta_{i,l}) = \max_{a_i \in A_i} \Big\{ \sum_{is \in IS} b_{i,l}(is) ER_i(is, a_i) + \gamma \sum_{o_i \in \Omega_i} P(o_i | a_i, b_{i,l}) V(\langle SE_{\theta_i}(b_{i,l}, a_i, o_i), \hat{\theta}_i \rangle) \Big\} \quad (3.9)$$

where $ER_i(is, a_i) = \sum_{a_j} R_i(is, a_i, a_j) Pr(a_j | \theta_{j,l-1})$, which denotes the expected reward for an interactive state and action of agent i.

With the value function in Equation 3.9, the optimal action, a_i^*, for the optimality criterion of infinite horizon with discounting, belongs to the set of optimal actions, $OPT(\theta_i)$:

$$OPT(\theta_{i,l}) = \arg\max_{a_i \in A_i} \Big\{ \sum_{is \in IS} b_{i,l}(is) ER_i(is, a_i) + \gamma \sum_{o_i \in \Omega_i} P(o_i | a_i, b_{i,l}) V(\langle SE_{\theta_i}(b_{i,l}, a_i, o_i), \hat{\theta}_i \rangle) \Big\} \quad (3.10)$$

3.3 Particle Filter

Particle filters (41) or sequential Monte Carlo (SMC) (42) methods are a set of sampling algorithms to approximate the posterior distributions of the states given observations in a Markov process . At each time step, a particle filter draws samples (or particles) from a proposal distribution, commonly the conditional distribution $p(s_t | s_{t-1})$ of the current state s_t given the previous s_{t-1}, then uses the observation function $p(o_t | s_t)$ to compute importance weights for all particles and resamples them according to the weights.

In single-agent POMDP settings, the belief update is usually achieved by using *Bayes filter* (13) in a two-step process, which corresponds to the decomposition of Equation 3.2:

- Prediction: to update agent's belief after performing an action a_i^{t-1}.

$$\hat{b}_i^t(s^t) = \sum_{s^{t-1} \in S} T_i(s^{t-1}, a_i^{t-1}, s^t) b_i^{t-1}(s^{t-1}) \tag{3.11}$$

- Correction: to correct the intermediate belief after receiving an observation o_i^t.

$$b_i^t(s^t) = \alpha O_i(s^t, a_i^{t-1}, o_i^t) \hat{b}_i^t(s^t) \tag{3.12}$$

When the system dynamic is nonlinear and non-Gaussian, and / or the state space is high-dimensional, Bayes filters are usually implemented as a particular particle filter called *bootstrap filter* (43; 9). The bootstrap filter uses the state transition function as the proposal distribution and updates the weights using the observation function. In Algorithm 2, it starts from an initial set of N particles, $s^{(n),t-1}$, which approximately represent the prior belief distribution, then propagates them through the transition function, $T_k(s^t | s^{(n),t-1}, a^{t-1})$, and lastly adjusts the weights according to the observation function, $O(o^t | s^{(n),t-1}, a^{t-1})$. After normalizing all the weights, the algorithm resamples from the intermediate particles and return the results as an approximation of the updated belief distribution.

Algorithm 2: Particle Filter for POMDPs

1 b^t = ParticleFilter $(b^{t-1} = s^{(n),t-1}, a^{t-1}, o^t)$

2 for all $s^{(n),t-1} \in b^{t-1}$:

3 Sample states $s^{(n),t} \sim T_k(s^t | s^{(n),t-1}, a^{t-1})$

4 Weight particles $s^{(n),t}$ with $w^{(n),t} = O(o^t | s^{(n),t-1}, a^{t-1})$

5 $\tilde{b}_t^{temp} =< s^{(n),t}, w^{(n),t} >$

6 Normalize weights $w^{(n),t}$, $\sum_{n=1}^{N} w^{(n),t} = 1$

7 Resample from $\tilde{b}_{k,l}^{temp}$ according to normalized $w^{(n),t}$

8 return $\tilde{b}^t = \{s^{(n),t}, n = 1, 2, ...N\}$

Let us illustrate how the particle filter works in a single-agent tiger problem (1). In a tiger problem, there are two identical doors, behind each of them lies a tiger or a pile of gold. If the tiger is behind the left door (TL), the gold is obviously behind the other one, and vice versa (i.e. TR). The agent can choose to listen (L) for the growl of the tiger from left (GL) or from right (GR), or open either the left door (OL) or the right door (OR) to get a big reward or a painful penalty. A particular setting of this game can be as follows: listening does not change tiger's location ($p(TL) = 1.0$ afterwards), opening any door will reset the tiger's location with equal probability ($p(TL) = 0.5$). The agent's observation accuracy with respect to the tiger's growl is relatively high (0.85). The listening action has a reward

of -1, opening the door with the tiger has a penalty of -100, and opening the door with the gold has a reward of 10.

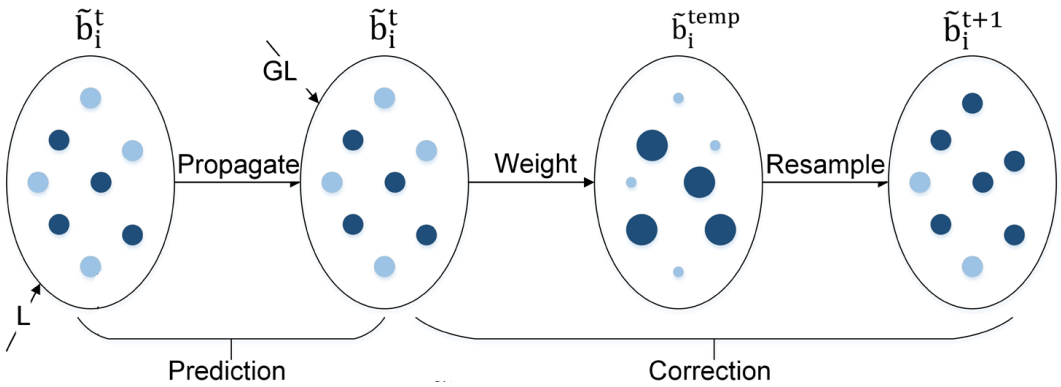


Figure 1: Particle filter for single-agent belief update in the tiger problem. Darker dots represent TL while lighter dots represent TR. Only 8 particles are used for the illustrative purpose.

Assume that the agent has a uniform prior belief about the tiger's location, namely p(TL)=0.5, and uses 8 particles to approximate the belief distribution, as shown in Figure 1. In the beginning, there are 4 darker particles and 4 lighter ones in the leftmost oval which correspond to the agent's prior belief. After an listening (L) action, the PF will propagate these 8 particles forward using the state transition function $T_i(s^{t-1}, a_i^{t-1}, s^t)$, and in this case all particles remain the same according to the transition function (in which listening does not change tiger's location). After getting an growl from the left (GL), the darker particles, which represent the probability of TL, will become bigger since they carry "heavier" weights

(i.e. 0.85). Computing the weights is achieved by utilizing the observation function $O_i(s^t, a_i^{t-1}, o_i^t)$. Lastly, the PF resamples all 8 particles according to their weights. Thus, in the rightmost oval, we get 6 particles for TL and 2 particles for TR, which approximately represent the agent's posterior belief.

3.4 Deep Reinforcement Learning

Reinforcement learning (RL) aims to have agents take actions in an environment so as to maximize some notion of cumulative reward. RL is essentially a sequential decision making problem, in which the reward signals are usually known and the cumulative reward, R, is assumed to be the sum of future discounted rewards (for infinite time horizon):

$$\begin{aligned} R &= r_0 + \gamma r_1 + \gamma^2 r_2 + ... \\ &= \sum_{t=0}^{\infty} \gamma^t r_t \end{aligned} \tag{3.13}$$

where r_t is the immediate reward at step t, $\gamma \in [0, 1]$ is the discounting factor.

Deep reinforcement learning (DRL) combines reinforcement learning (RL) and deep learning together, aiming to solve complex problems in sequential decision making. Between the two combined approaches, the former usually defines the objective of a learning task while the latter provides the mechanism. Many successes DRL works scaled up prior works in RL to high-dimensional problems, in which low dimensional feature representations are usually learned using neural networks (44). Unlike traditional parametric methods, DRL can deal efficiently with the curse of dimensionality (45). For instance, convolutional neural networks (CNNs) can be used to learn directly from raw, high dimensional visual inputs.

Similarly to RL, DRL has two major approaches, model-free and model-based, depending on whether the world model is given or the actual design of the network (which is usually due to the complexity of the problem).

3.4.1 Model-Free Deep Reinforcement Learning

In a model-free DRL approach, an end-to-end neural network is usually used to learn a policy, which maps an agents' history of observations and actions to an action. The goal of this approach is to represent a control policy that has good long-term behaviors, normally quantified as minimizing a sequence of time-dependent costs, e.g. the sum of future discounted rewards defined in Equation 3.13. The neural network architectures used in most recent model-free DRL works typically consist of convolutional layers for feature extraction, recurrent layers to estimate the underlying state of the environment, and fully connected layers to map the estimated state to an action(31; 54).

In the context of partial observability, the agent's action selection is modeled as a mapping between the history space, H, and the action space, A, which is called policy:

$$\pi : H \rightarrow A \tag{3.14}$$

where a particular history $h_t =< a_0, o_1 >,< a_1, o_2 >, ..., < a_{t-1}, o_t >$, t is the time step. The action prescribed by the policy π at time t would be $a_t = \pi(h_t)$.

Given the definition of a policy, we can evaluate its quality by defining a value function $Q^\pi(h, a)$, representing the value of taking action a given history h. On the contrast, as we will see in Section 3.4.2, the value function in model-based DRL is represented as a function of belief and action $Q^\pi(b, a)$.

The model-free DRL approach usually estimates this value function $Q(h, a)$ using a nonlinear function approximator, such as a deep neural network. Namely, the Q values can be learned by a neural network that is characterized by weights and biases collectively denoted as w. The inputs of this neural network are histories and actions, and outputs are optimal actions. Therefore, the problem of learning the Q value becomes a parameter optimization problem of minimizing the following loss function:

$$\mathcal{L}(w_i) = \mathop{\mathbb{E}}_{h,a\sim p(h,a)}[(r + \gamma \mathop{max}_{a} Q(h', a'|w_i) - Q(h, a|w_i)]^2 \tag{3.15}$$

where $p(h, a)$ is the behavior distribution over histories and actions that is usually defined by the training algorithm.

So far we have not given any intuitions on the reason of choosing this particular loss function or the way of minimizing it, because conventionally the definition of optimal value function Q^*, which is the inspiration of this loss, is often given on the belief space (i.e. $Q^*(b, a)$ instead of $Q^*(h, a)$). The $Q(h, a)$ is used in Equation 3.15 as an alternative to $Q(b, a)$ since there is no belief computation in model-free DRL due to the lack of model functions. Hence, we will give more insights and explanations on deriving this loss function in the following Section 3.4.2.

Despite recent advances in model-free DRL (31; 54), model-free networks are inherently reactive and lack explicit planning computation. The success of model-free DRL is largely due to the learning algorithm, which essentially learns a reactive policy to select actions associated with good long-term rewards (29).

3.4.2 Model-Based Deep Reinforcement Learning

In general, model-based DRL uses neural networks to approximate one or more of the following important components in a sequential decision making task: the value function $Q(b, a; w)$, the policy $\pi(b; w)$, and the model (e.g. the state transition $T(s'|s, a; w)$, observation $O(o|s', a; w)$ and reward functions $R(r|s, a; w)$ for a POMDP), where b, s, a and o represent the belief, state, action and observation respectively, and the parameters w are the weights in a DNN (46; 47).

In the context of partial observability, the agent's action selection is modeled as a mapping between the belief state space, B, and the action space, A, which is called policy:

$$\pi : B \rightarrow A \tag{3.16}$$

The policy provides an optimal action $a \in A$ when in belief state $b \in B$.

The optimal action-value function $Q^*(b, a)$ can be defined as the maximum expected return achievable by following any strategy given the current belief $b_t = b$ and action $a_t = a$ at time t:

$$Q^*(b, a) = \max_{\pi} \mathbb{E}[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}] \tag{3.17}$$

where T is the time horizon and $T = \infty$ if it is infinite.

In the context of partial observability, the optimal action-value function satisfies the Bellman equation. Intuitively, if the optimal value $Q^*(b', a')$ of the subsequent belief b' was known for all possible

actions a', then the optimal strategy is to choose the action a' that maximizes the expected value of $r + \gamma \, max_{a'} \, Q^*(b', a')$:

$$Q^*(b, a) = \mathop{\mathbb{E}}_{b'}[r + \gamma \mathop{max}_{a'} Q^*(b', a')|b, a] \tag{3.18}$$

Using Equation 3.18 as an iterative update, it becomes the value iteration (VI) algorithm:

$$Q_{i+1}(b, a) = \mathop{\mathbb{E}}_{b'}[r + \gamma \mathop{max}_{a'} Q_i(b', a')|b, a] \tag{3.19}$$

The Q in Equation 3.19 converges to the optimal value function, $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$, for infinite time horizon.

However, the size of the joint space between belief and action $|B \times A|$ is usually huge in practical problems. In model-based DRL, a neural network characterized by weights w is used to estimate the action-value function, namely $Q^*(b, a) \approx Q^*(b, a|w)$. The parameter space is usually smaller than the aforementioned joint space, namely $|W| << |B \times A|$. The inputs of this neural network are beliefs and actions, and outputs are optimal actions. Instead of updating each Q value, we can update the parameters of the neural network by minimizing the following loss function:

$$\mathcal{L}(w_i) = \mathop{\mathbb{E}}_{b,a \sim p(b,a)}[(r + \gamma \mathop{max}_{a} Q(b', a'|w_i) - Q(b, a|w_i)]^2 \tag{3.20}$$

where $p(b, a)$ is the behavior distribution over beliefs and actions that is usually defined by the training algorithm.

To learn the models of the task environment and use learned models to plan, model-based DRL is usually divided into two phases: the learning phase and the planning phase. In the learning phase, the model functions are unknown or assumed unknown (i.e. the network starts from random weights) and the reinforcements / reward signals are observable; in the planning phase, the reinforcements are unobservable and the learned models are used to compute the policy. One of the intuition behind this approach is to learn the model functions which allow for simulations of the real environment without interacting with it directly. Since direct iterations with the real environment may be limited in practice, learning the models in this RL approach reduces the required interactions with reality (44).

To embed model learning and the planning computation in a single neural network, the main idea is to represent matrix summations and multiplications (i.e. linear operations) by convolution layers and maximum operations by max-pooling layers. Specifically, for a POMDP model $POMDP(w) =< S, A, \Omega, T(\cdot|w), O(\cdot|w), R(\cdot|w) >$, the belief update can be implemented as a convolutional layer, and each channel in the convolution layer corresponds to the belief update for a specific action.

$$\hat{b}_j^t(s^t) = \sum_{s^{t-1} \in S} W_a b_j^{t-1}(s^{t-1}) \tag{3.21}$$

$$b_j^t(s^t) = \alpha O(s^t, a_j^{t-1}, o_j^t) \hat{b}_j^t(s^t) \tag{3.22}$$

where W_a are the weights of the neural network, denoting the probabilities in the state transition function $T(s'|s, a)$. The observation function $O(s^t, a_j^{t-1}, o_j^t)$ will be used to correct the belief prediction result in Equation 3.21 by making a dot product between $O(s^t, a_j^{t-1}, o_j^t)$ and $\hat{b}_j^t(s^t)$. The observation function $O(s^t, a_j^{t-1}, o_j^t)$ itself can be a convolution tensor with each channel representing a particular observation.

In the planning computation, each iteration of the value iteration (VI) algorithm can be viewed as passing the previous reward and value through a convolutional layer and then a max-pooling layer. In this analogy, each channel in the convolutional layer corresponds to the Q-value function for a particular action, and kernel / filter weights of the convolutional layer correspond to the transition probabilities. Therefore, K iterations of value iteration has been performed after recurrently applying the convolutional layer K times.

$$Q_{k+1}(b,a) = R(b,a) + \gamma \sum_{b'} W_a V_k(b') \tag{3.23}$$

$$V_k(b) = \max_a Q_k(b,a) \tag{3.24}$$

where W_a are the weights of the neural network, denoting the probabilities in the transition function in the belief state space $T^b(b'|b,a)$, and max_a is now a max-pooling layer over actions a. Notice that $R(b,a) = \sum_s b(s)R(s,a)$ and $\sum_{b'} T^b(b'|b,a) = \sum_o Pr(o|b,a)$.

Unlike model-free DRL, this particular model-based approach learns the task model as well as an approximate planning computation for solving the task. Usually such a model-based computation has better generalization capability in a diverse set of tasks (29; 10). As we are interested in multi-agent sequential decision making, we prefer a model-based approach. Since intuitively other agents' models generate their actions which indirectly impact our actions, having a model of others almost surely benefits our decision making.

Practically, during the learning phase, to prevent divergence when optimizing the network parameters, many works have adopted three major techniques to stabilize the learning process (10; 31). First, samples of experience replay tuples $[b, a, r, b']$ are used in batches to minimize the loss function, which

breaks the dependence in data as they are generated sequentially. Second, adaptive optimizers, such as Adam (49) and RMSProp (50), other than simple stochastic gradient descent (SGD), are widely used to regulate the parameter learning rate. These adaptive optimizers can accelerate the learning process and often lead to lower gradients. Third, a separate target network is normally used to provide updates to the main network, meaning that the parameters from the previous iteration, w_{i-1}, are held fixed when optimizing the loss function $\mathcal{L}(w_i)$ for the current iteration.

CHAPTER 4

INTERACTIVE BELIEF UPDATE USING SEQUENTIAL MONTE CARLO SAMPLING

In Chapter 3.2, we introduced the I-POMDP decision making framework we used for a multi-agent planning task. We see that due to the complexity of the augmented interactive belief space, exact belief update in I-POMDPs is computationally intractable. To plan and act in such an environment, we have adopted a customized sequential Monte Carlo sampling methods for I-POMDPs. In this chapter, we introduce the sampling method that we have customized: the interactive particle filter (I-PF) that has been extended to the intentional model space.

4.1 Interactive Particle Filter over Intentional Model Space

The interactive particle filter (I-PF) generalizes the particle filter (PF) to the multi-agent domain, serving as a filtering algorithm for the interactive belief update of I-POMDPs (8). In I-PF, the state transition function is used as the proposal (candidate) distribution, usually called the bootstrap filter (43), and both agents' observation functions are used to correct the propagated belief. However, due to the enormous interactive belief space, the I-PF implementation has a strong assumption that the other agent's intentional model except his belief, i.e. j's frame $\hat{\theta}_j$, is known. Thus, the I-PF reduces I-POMDP's interactive belief update from a complete joint space of state and others' models, $S \times \Theta_{j,l-1}$, to a significantly smaller space of state and others' beliefs, $S \times B_{j,l-1}$.

Our interactive belief update algorithm, described in Algorithm 3 and 4, however, generalizes I-PF to larger intentional model space which contains other agents' beliefs, nesting levels, and transition, observation and reward functions. In the remaining part of this chapter, we will firstly give a brief introduction of our algorithm and discuss the motivations for each sampling step. Then we will show the major differences between our algorithm and the I-PF, since this generalization is nontrivial. A concrete example is given in Figure 3 in Chapter 5 using the two-agent version of the tiger problem.

Algorithm 3: Interactive Belief Update

$\tilde{b}^t_{k,l}$ = InteractiveBeliefUpdate($\tilde{b}^{t-1}_{k,l}$, a^{t-1}_k, o^t_k, $l > 0$)

For $is^{(n),t-1}_k =< s^{(n),t-1}, \theta^{(n),t-1}_{-k,l-1} > \in \tilde{b}^{t-1}_{k,l}$:

  sample $a^{t-1}_{-k} \sim P(a_{-k}|\theta^{(n),t-1}_{-k,l-1})$

  sample $s^{(n),t} \sim T_k(s^t|s^{(n),t-1}, a^{t-1}_k, a^{t-1}_{-k})$

  for $o^t_{-k} \in \Omega_{-k}$:

    if $l = 1$:

      $b^{(n),t}_{-k,0}$ = Level0BeliefUpdate($\theta^{(n),t-1}_{-k,0}$,a^{t-1}_{-k}, o^t_{-k})

      $is^{(n),t}_k =< s^{(n),t}, \theta^{(n),t}_{-k,0} >$

    else:

      $b^{(n),t}_{-k,l-1}$ = InteractiveBeliefUpdate($\tilde{b}^{(n),t-1}_{-k,l-1}$, a^{t-1}_{-k}, o^t_{-k}, $l-1$)

      $\theta^{(n),t}_{-k,l-1} =< b^{(n),t}_{-k,l-1}, \hat{\theta}^{(n),t-1}_{-k,l-1} >$

      $is^{(n),t}_k =< s^{(n),t}, \theta^{(n),t}_{-k,l-1} >$

    $w^{(n)}_t = O^{(n)}_{-k}(o^t_{-k}|s^{(n),t}, a^{t-1}_k, a^{t-1}_{-k})$

    $w^{(n)}_t = w^{(n)}_t \times O_k(o^t_k|s^{(n),t}, a^{t-1}_k, a^{t-1}_{-k})$

    $\tilde{b}^{temp}_{k,l} =< is^{(n),t}_k, w^{(n)}_t >$

normalize all $w^{(n)}_t$ so that $\sum_{n=1}^{N} w^{(n)}_t = 1$

resample from $\tilde{b}^{temp}_{k,l}$ according to normalized $w^{(n)}_t$

resample $\theta^{(n),t}_{-k,l-1} \sim N(\theta^t_{-k,l-1}|\theta^{(n),t-1}_{-k,l-1}, \Sigma)$

return $\tilde{b}^t_{k,l} = is^{(n),t}_k =< s^{(n),t}, \theta^{(n),t}_{-k,l-1} >$

The Algorithm 3 requires inputs of the modeling agent's prior belief, $\tilde{b}_{k,l}^{t-1}$, which is represented as a set of n samples $is_k^{(n),t-1}$, along with the action, a_k^{t-1}, the observation, o_k^t, and the belief nesting level, $l > 0$. Here k represents either agent i or j, and $-k$ represents the other agent, j or i, correspondingly. We assume that the modeled agent's action set A_{-k}, observation set Ω_{-k} and optimality criteria OC_k are known to all agents. We want to learn the other agent's initial belief about the physical state, b_{-k}^0, the transition function, T_{-k}, the observation function, O_{-k} and the reward function, R_{-k}.

The initial belief samples, $is_k^{(n),t-1}$, are generated from the prior nested belief in the similar way as described in the I-PF literature (8) except that $T_{-k}^{(n)}, O_{-k}^{(n)}$, and $R_{-k}^{(n)}$ are sampled from their prior distributions as well. Notice that $T_{-k}^{(n)}, O_{-k}^{(n)}$, and $R_{-k}^{(n)}$ are all part of the frame, namely $\hat{\theta}_{-k}^{(n)} =< A_{-k}, \Omega_{-k}, T_{-k}^{(n)}, O_{-k}^{(n)}, R_{-k}^{(n)}, OC_k >$, as appeared in line 7 and 10 in Algorithm 3.

With initial belief samples, Algorithm 3 starts from propagating each sample forward in time and computing their weights (line 1-15), then it resamples according to the weights and similarity between models (line 16-18). Intuitively, the samples associated with actual observations perceived by agent k will gradually carry larger weights and be resampled more often, therefore they will approximately represent the actual belief. Specifically, for each of $is_k^{(n),t-1}$, the algorithm samples the other agent's optimal actions a_{-k}^{t-1} given its model from $P(a_{-k}|\theta_{-k}^{(n),t-1})$ (line 2), which can be solved by any POMDP solver. Then it samples the physical state $s^{(n),t}$ using the state transition function $T_k(s^t|s^{(n),t-1}, a_k^{t-1}, a_{-k}^{t-1})$ (line 3). Then for each possible observation, if the current nesting level l is 1, it calls the 0-level belief update, described in Algorithm 4, to update other agents' beliefs over physical state $b_{-k,0}^t$ (line 5 to 8); or, if l is greater than 1, it recursively calls itself at a lower level $l-1$ (line 9 to 11). Weights of the samples, $w_t^{(n)}$, are computed using both agents' observation likelihoods (line 12,

13). Lastly, the algorithm normalizes all the weights (line 15) and resamples the intermediate samples according to the weights (line 16). Using a Gaussian distribution, the algorithm resamples again from similar neighboring models to avoid divergence due to the high dimensionality of the belief space (line 17).

Algorithm 4: Level-0 Belief Update

$b_{k,0}^t$ =Level0BeliefUpdate$(\theta_{k,0}^{t-1}, a_k^{t-1}, o_k^t)$

1 get T_k and O_k from $\theta_{k,0}^{t-1}$

2 $P(a_{-k}^{t-1}) = 1/|A_{-k}|$

3 for $s^t \in S$:

4 for s^{t-1}:

5 for $a_{-k}^{t-1} \in A_{-k}$:

6 $P(s^t|s^{t-1}, a_k^{t-1}) += T_k(s^t|s^{t-1}, a_k^{t-1}, a_{-k}^{t-1})P(a_{-k}^{t-1})$

7 $sum += P(s^t|s^{t-1}, a_k^{t-1})b_{k,0}^{t-1}(s^{t-1})$

8 for $a_{-k}^{t-1} \in A_{-k}$:

9 $P(o_k^t|s^t, a_k^{t-1}) += O_k(o_k^t|s^t, a_k^{t-1}, a_{-k}^{t-1})P(a_{-k}^{t-1})$

10 $b_{k,0}^t = sum \times P(o_k^t|s^t, a_k^{t-1})$

11 normalize and return $b_{k,0}^t$

The 0-level belief update, described in Algorithm 4, takes agent model, $\theta_{k,0}^{t-1}$, action, a_k^{t-1}, and observation, o_k^t, as input arguments and returns the belief about the physical state, $b_{k,0}^t$. The other agent's actions are treated as uniform noise (line 2), and transition and observation functions are passed in within the first input argument $\theta_{k,0}^{t-1}$. Line 6 averages over all possible actions of the other agent, a_{-k}^{t-1}, to computes the true state transition function by weighting the original transition, T_k, with the probability of the other agent taking each action, $P(a_{-k}^{t-1})$. Similarly, line 9 computes the actual observation function by weighting the original observation function, O_k, with the probability of the other agent taking each action, $P(a_{-k}^{t-1})$. Finally, it returns the normalized belief $b_{k,0}^t$. Note that both the transition function, $T_k(s^t|s^{t-1}, a_k^{t-1}, a_{-k}^{t-1})$, and the observation function, $O_k(o_k^t|s^t, a_k^{t-1}, a_{-k}^{t-1})$, contained in the agent's intentional model, θ_k^{t-1}, depend on model parameters of the particular agent on the 0th level.

Our interactive belief update algorithm differs in three major ways from the I-PF. First, in order to update the belief over intentional model space of other agents, their initial belief, nesting level, transition function, observation function and reward function in their frames are all unknown and become samples. For instance, the set of n samples of other agents' intentional models $\theta_{-k,l-1}^{(n),t-1} = < b_{-k,l-1}^{(n),t-1}$, $A_{-k}, \Omega_{-k}, T_{-k}^{(n)}, O_{-k}^{(n)}, R_{-k}^{(n)}, OC_k >$. The observation function of the modeled agents, $O_{-k}^{(n)}$ $(o_{-k}^t|s^{(n),t}$, a_k^{t-1}, $a_{-k}^{t-1})$ in line 12 of Algorithm 3, is now randomized consequently. Second, the transition and observation functions of the level-0 agent, in line 6 and 9 of Algorithm 4, are passed in as input arguments which correspond to each model sample. Lastly, we add another resampling step in line 18 to avoid divergence, by resampling the model samples from a Gaussian distribution with the mean of current sample value. This additional resampling step is nontrivial, since empirically the samples diverge quickly due to the drastically enlarged sample space.

4.2 Illustration Using Two-Agent Tiger Problem

We illustrate how the algorithms work in the multi-agent tiger problem (5). The two-agent tiger problem is a generalization of the classic single agent tiger problem with additional observations caused by others agents' actions. Also, the transition function, observation function and reward function all contain others' actions.

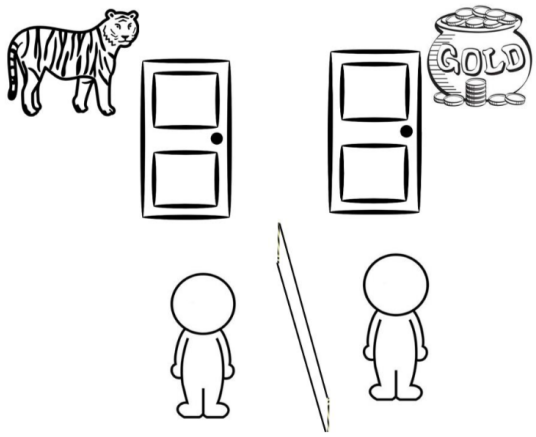


Figure 2: The two agent tiger problem. Agents can not directly observe the other's actions.

Specifically, there are two identical doors, behind each of them lies a tiger or a pile of gold. The tiger can be behind the left door (TL), while the gold is behind the other one, and vice versa (i.e. TR). Both of the agents can choose to listen (L) for the growl of the tiger from left (GL) or right (GR), along with additional observations of creak from left (CL) or right (CR) caused by the other agents' actions. They can also choose to open either the left door (OL) or right door (OR) to get a big reward or a painful

penalty. The agents are unable to directly observe the other's actions, instead, they try to infer them based on their extended observations. We assume there are two agents i and j, i's model is known and shown in Table Table I, but j's model is unknown and we want to learn it through i's observations of the environment.

Let us define the two-agent tiger problem in the form of an interactive POMDP, which is defined as a six tuple of agent i as $I\text{-}POMDP_i = \langle IS_{i,l}, A, \Omega_i, T_i, O_i, R_i \rangle$.

- $IS_{i,1} = S \times \theta_{j,0}$, $S = \{$tiger on the left (TL), tiger on the right (TR)$\}$,
- $A = A_i \times A_j$: $\{$listen (L), open the left door (OL), open the right door (OR)$\}\times\{$ listen (L), open the left door (OL), open the right door (OR)$\}$,
- Ω_i: $\{$growl from left (GL), growl from right (GR)$\}$ $\times$ $\{$creak from left (CL), creak from right (CR), silence (S)$\}$,
- $T_i = T_j : S \times A_i \times A_j \times S \rightarrow [0, 1]$ now becomes a joint state transition probability that involves both agents' actions, the tiger's position is reset to the left or the right door with $0.5/0.5$ whenever an agent opens the door, and remains with 1.0 probability when they both listen.
- $O_i : S \times A_i \times A_j \times \Omega_i \rightarrow [0, 1]$ becomes a joint observation probability that involves both agents' actions. The observation accuracy of agent i is the probability of hearing a growl (0.85) (if tiger is on the left) times the probability of hearing a creak (0.9) (if the other agent opens a door).
- $R_i : IS \times A_i \times A_j \rightarrow R_i$. Agent i gets -1 when he listens, -100 when he opens the tiger door, or 10 when he opens the gold door. The rewards are usually independent of j's actions.

TABLE I: Transition, observation and reward functions of agent i at level l.

(a) Transtion function T_i

S	A_i, A_j	$p(TL)$	$p(TR)$
TL	L, L	1.0	0.0
TR	L, L	0.0	1.0
$*$	$OL, *$	0.5	0.5
$*$	$OR, *$	0.5	0.5
$*$	$*, OL$	0.5	0.5
$*$	$*, OR$	0.5	0.5

(b) Reward function R_i

S	A_i, A_j	R
$*$	$L, *$	-1
TL	$OL, *$	-100
TR	$OR, *$	-100
TL	$OR, *$	10
TR	$OL, *$	10

(c) Observation function O_i

S	A_i, A_j	$p(GL, CL)$	$p(GL, CR)$	$p(GL, S)$	$p(GR, CL)$	$p(GR, CR)$	$p(GR, S)$
TL	L, L	$0.85 * 0.05$	$0.85 * 0.05$	$0.85 * 0.9$	$0.15 * 0.05$	$0.15 * 0.05$	$0.15 * 0.9$
TR	L, L	$0.15 * 0.05$	$0.15 * 0.05$	$0.15 * 0.9$	$0.85 * 0.05$	$0.85 * 0.05$	$0.85 * 0.9$
TL	L, OR	$0.85 * 0.05$	$0.85 * 0.9$	$0.85 * 0.05$	$0.15 * 0.05$	$0.15 * 0.9$	$0.15 * 0.05$
TR	L, OR	$0.15 * 0.05$	$0.15 * 0.9$	$0.15 * 0.05$	$0.85 * 0.05$	$0.85 * 0.9$	$0.85 * 0.05$
TL	L, OL	$0.85 * 0.9$	$0.85 * 0.05$	$0.85 * 0.05$	$0.15 * 0.9$	$0.15 * 0.05$	$0.15 * 0.05$
TR	L, OL	$0.15 * 0.9$	$0.15 * 0.05$	$0.15 * 0.05$	$0.85 * 0.9$	$0.85 * 0.05$	$0.85 * 0.05$
$*$	$OR, *$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$
$*$	$OL, *$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$

From Equation 3.7 we know that when the nesting level bottoms down to 0, the modeled agent reduces to a POMDP agent. Assume that j is modeled as a POMDP agent at level 0, although j's model is unknown, we can understand what the parameterized model functions look like using a particular example in Table II. We see that seven parameters are used to represent the transition, observation and reward functions, and one additional parameter can be used to represent the initial belief. Namely, the model of a particular agent j is $\theta_j = \langle b_j(s) = 0.5, p_{T_1} = 0.67, p_{T_2} = 0.5, p_{O_1} = 0.85, _{O_2} = 0.5, p_{R_1} = -1, p_{R_2} = -100, p_{R_3} = 10\rangle$, where $b_j(s)$ is j's initial belief, p_{T_1} and p_{T_2} parametrize j's transition function, p_{O_1} and p_{O_2} parametrize j's observation function, and p_{R_1}, p_{R_2} and p_{R_3} parametrize j's reward function. There can be one additional parameter to use if the nesting level is being modeled as well. We will look into more details about parameterization of other agents' model functions in Chapter 5.1.

TABLE II: Transition, observation and reward functions of a particular agent j at level 0.

(a) Transtion function T_j

S	A	p(TL)	p(TR)
TL	L	**0.67**	0.33
TR	L	0.33	0.67
*	OL	**0.5**	0.5
*	OR	0.5	0.5

(b) Observation function O_j

S	A	p(GL)	p(GR)
TL	L	**0.85**	0.15
TR	L	0.15	0.85
*	OR	0.5	0.5
*	OL	**0.5**	0.5

(c) Reward function R_j

S	A	R
*	L	**-1**
TL	OL	**-100**
TR	OR	-100
TL	OR	**10**
TR	OL	10

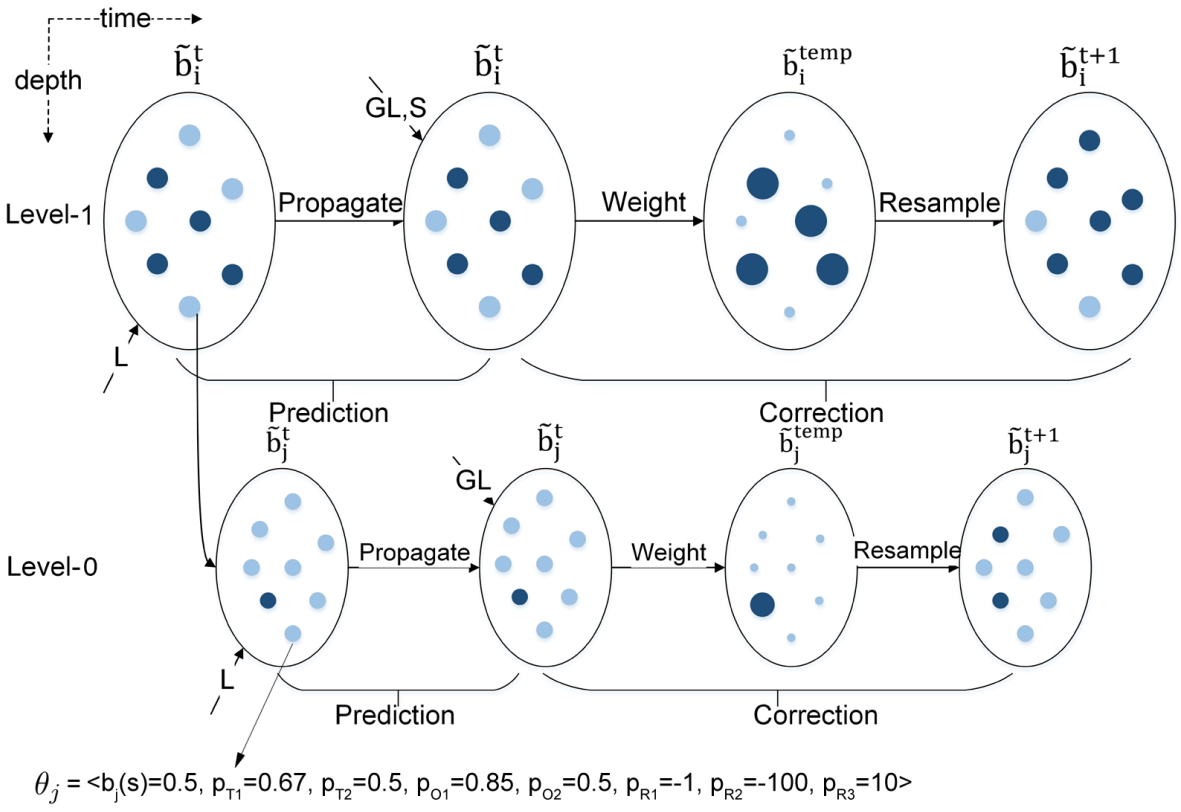


Figure 3: An illustration of interactive belief update algorithm over intentional model space using two-agent tiger problem with level-1 nesting.

We illustrate the interactive belief update algorithm using the two-agent tiger problem. Suppose there are two agents, i and j, in this tiger problem, the nesting (strategy) level is 2 and the sample size is 8. In Figure 3, the subscripts denote corresponding agents and each dot represents a particular belief particle. The propagation step is implemented in lines 2 to 11 in Algorithm 3, the weighting step is in lines 12 to 15, and the resampling step is in lines 16 and 17. The "plain" belief for a particular level-

0 model sample $\theta_j = \langle 0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10 \rangle$ is updated using Algorithm 4, and the optimal action for this bottom-level POMDP agent is given by executing the state-of-the-art SARSOP solver (51).

CHAPTER 5

LEARNING OTHERS' INTENTIONAL MODELS USING I-POMDPS

In this chapter, we discuss the model parameterization, briefly demonstrate the learning process, and analyze the solution quality, convergence and computational cost with further experiments on deeper nesting levels. We evaluate our algorithm on the multi-agent tiger problem and UAV reconnaissance problem (8). The multi-agent tiger game is a generalization of the classic single agent tiger game which has been discussed in previous chapters. The UAV reconnaissance problem contains a 3x3 grid in which the agent (UAV) tries to capture the moving target while the reconnaissance target tries to reach to the safe house. The UAV is unaware of which location contain the target, but it is aware of its own location. The UAV may receive a noisy observation which provides the locations that likely contains the target. The UAV can move in any of the four cardinal directions to the adjacent locations (so as the target), or stay in the current location for listening for more information. The target can sense the danger of being spotted by the UAV with high uncertainty, when the UAV is in adjacent locations (8).

5.1 Model Parameterization

The initial step of solving an I-POMDP in our approach is to parameterize other agents' models in terms of an I-POMDP or POMDP, depending on the modeling agent's strategy level. Then the model parameters can be sampled and updated using the interactive belief update algorithm discussed in Chapter 4 for solving the planning task.

Here we give an example of the parameterization using the two-agent tiger problem. For the sake of brevity, we assume there are two agent i and j, and the strategy level is 1, but the sampling algorithm can be extended to more than two agents and higher nesting levels in a straightforward manner. Notice that we show experimental results with higher nesting levels in Chapter 5.2.

TABLE III: Unknown parameters for transition, observation and reward functions

(a) Transtion fucntion T_j

S	A	p(TL)	p(TR)
TL	L	p_{T1}	$1 - p_{T1}$
TR	L	$1 - p_{T1}$	p_{T1}
TL	OL	p_{T2}	$1 - p_{T2}$
TR	OL	$1 - p_{T2}$	p_{T2}
TL	OR	$1 - p_{T2}$	p_{T2}
TR	OR	p_{T2}	$1 - p_{T2}$

(b) Observation function O_j

S	A	p(GL)	p(GR)
TL	L	p_{O1}	$1 - p_{O1}$
TR	L	$1 - p_{O1}$	p_{O1}
TL	OL	p_{O2}	$1 - p_{O2}$
TL	OL	$1 - p_{O2}$	p_{O2}
TL	OR	$1 - p_{O2}$	p_{O2}
TR	OR	p_{O2}	$1 - p_{O2}$

(c) Reward R_j

S	A	R
*	L	r_{R1}
TL	OL	r_{R2}
TR	OR	r_{R2}
TL	OR	r_{R3}
TR	OL	r_{R3}

For the two-agent tiger problem, what we want to learn is over the entire intentional model space of agent j: $\theta_j =< b_j(s), A_j, \Omega_j, T_j, O_j, R_j, OC_j >$. As mentioned before we assume that A_j and Ω_j are known, and OC_j is infinite horizon with discounting. We want to recover the possible initial belief

b_j^0 about the physical state, the transition, T_j, the observation, O_j, and the reward, R_j. Thus, the main idea of our experiment is to make Bayesian parametric learning with the help of the interactive belief sampling algorithm discussed in Chapter 4. In terms of a level-0 POMDP, what we really want to learn about the other agent j is:

- b_j^0: agent j's initial belief about the physical state of the environment.
- T_j: agent j's transition function, parametrized by p_{T_1} and p_{T_2} as shown in Table III(a).
- O_j: agent j's observation function, parametrized by p_{O_1} and p_{O_2} as shown in Table III(b).
- R_j: agent j's reward function, parametrized by p_{R_1}, p_{R_2} and p_{R_3} as shown in Table III(c).

We see that in Table III it is an enormous 8-dimensional parameter space to learn from: $b_j^0 \times p_{T1} \times p_{T2} \times p_{O1} \times p_{O2} \times p_{R1} \times p_{R2} \times p_{R3}$, where $\{b_j, p_{T1}, p_{T2}, p_{O1}, p_{O2}\} \in [0,1] \subset \mathbb{R}$, and $\{p_{R1}, p_{R2}, p_{R3}\} \in [-\infty, +\infty] \subset \mathbb{R}$.

5.1.1 Preliminary Results

For the two-agent tiger problem, we fix the number of samples to be 2000 and run experiments of learning three different models of agent j, as shown in Figure Figure 4:

1. $\theta_{j_1} :< b_j^0 = 0.5, p_{T_1} = 0.67, p_{T_2} = 0.5, p_{O_1} = 0.85, p_{O_2} = 0.5, p_{R1} = -1, p_{R2} = -100, p_{R3} = 10 >$
2. $\theta_{j_2} :< b_j^0 = 0.5, p_{T_1} = 1.00, p_{T_2} = 0.5, p_{O_1} = 0.95, p_{O_2} = 0.5, p_{R1} = -1, p_{R2} = -10, p_{R3} = 10 >$
3. $\theta_{j_3} :< b_j^0 = 0.5, p_{T_1} = 0.66, p_{T_2} = 0.5, p_{O_1} = 0.85, p_{O_2} = 0.5, p_{R1} = 10, p_{R2} = -100, p_{R3} = 10 >$

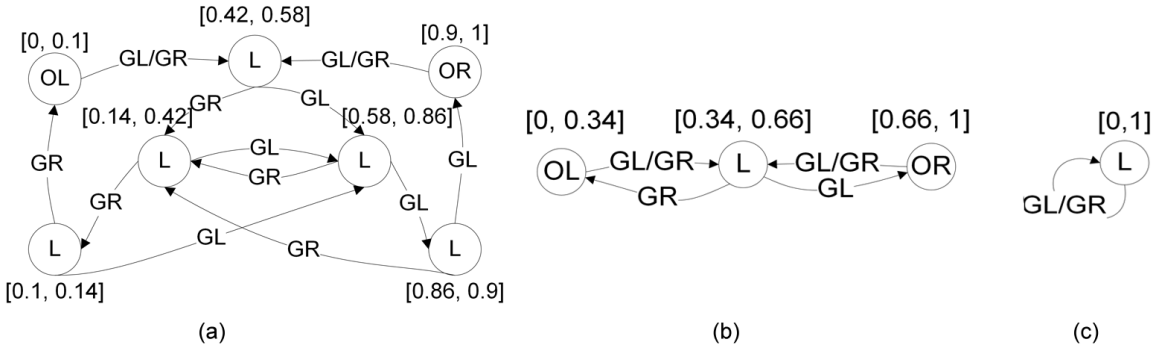


Figure 4: Optimal policies denoted as FSCs of: (a) θ_{j_1}=⟨0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10⟩, (b) θ_{j_2} =⟨ 0.5, 1, 0.5, 0.95, 0.5, -1, -10, 10⟩, and (c) θ_{j_3} =⟨0.5, 0.66, 0.5, 0.85, 0.5, 10, -100, 10⟩.

We compare the performance of three different modeling agents i, a level-1 I-POMDP, a level-2 I-POMDP and a subintentional model (fictitious play), in each experiment. We show results of learning models of the level-1 agent j whose policy is in Figure 4, and give an performance comparison later in Figure 8.

These three particular opponents are chosen to demonstrate the learning ability of our algorithm. The aim of first experiment is trying to learn models of agent j who is modeling his opponent using a subintentional model. As shown in Figure 4 (a), the actual policy of agent j's is to anticipate three continuous growls from a same direction and then open the opposite door. The second experiment involves agent j equipped with high listening accuracy of 0.95 and small penalty of -10 for encountering the tiger, i.e. the agent j alternately opens door and listens as shown in Figure 4 (b). And the third experiment involves a simple agent j who always listens since the listening penalty is now equal to the

reward, i.e. 10 as shown in Figure 4 (c). In conclusion, one can view the difficulties of learning such agents' models as relatively hard, medium, and easy, since the policy difficulties decrease in the three experiments. On the contrary, the parameters being learned will be less definite, since there are more possible models which can generate the same policy.

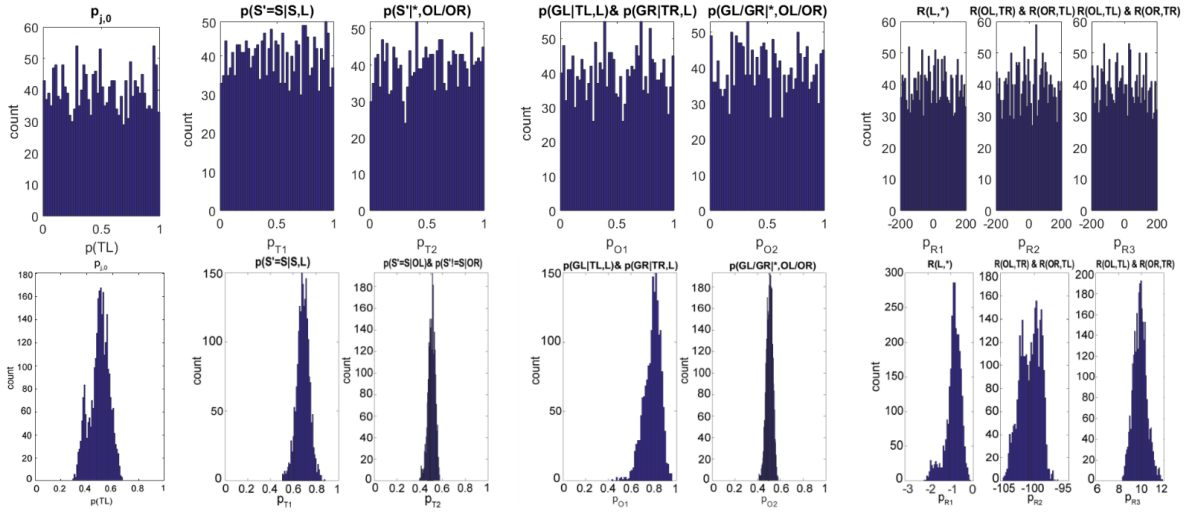


Figure 5: Histograms of assigned uniform priors (top row) and learned posteriors (bottom row) over model parameters θ_{j_1} $=\langle$0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10$\rangle$ in Figure 4 (a). The modes of the posteriors are close to the true model parameters.

For the first experiment, we want to learn a relatively complicated agent j, which has a subintentional model as $< 0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10 >$. This agent j assumes that the other agent's

(i.e. i's) actions are drawn from a uniform distribution with equal probability, and therefore is a no-information model. Accordingly, as shown in Figure 4 (a), the actual policy of agent j's is expecting three continuous growls from a same direction then opening the opposite door. Since there are uncertainties due to agent i's imperfect hearing ability (i.e. observation function) which makes the verification of our learning algorithm unclear, in the first experiment we simulated agent i's observations sequence in order to firstly verify the correctness of our algorithm. The simulated observation sequence consists of repeated three growls from the same direction and a creak caused by the other agent's door-opening action: {GL,S GL,S GL,S GL,CR GL,S GL,S GL,S GR,CR GL,S GL,S GL,S GR,CR GL,S GL,S GL,S GR,CR GR,S GR,S GR,S GR,CL GR,S GR,S GR,S GR,CL GR,S GR,S GR,S GR,CL GR,S GR,S GR,S GR,CL GR,S GR,S GR,S GR,CL GR,S GR,S GR,S GL,CL GL,S GL,S GL,S GR,CR GL,S GL,S GL,S GR,CR GR,S GR,S}

To exclude the potential impacts from informative prior belief distributions, we assign uninformative uniform priors to each parameter, which is shown in Figure 5. These uniform priors are: $\{b_j^0, p_{T1}, p_{T2}, p_{O1}, p_{O2}\} \sim U(0,1)$, $p_{R1}, p_{R2}, p_{R3} \sim U(-200, 200)$. In Figure 5, we give the posterior distribution over agent j's model parameters after 50 time steps. We only show the marginal distributions of each model parameter in terms of histograms, because the parameter space we are learning from is eight dimensional. Notice that most of the samples converge to the true parameter values of agent j's model.

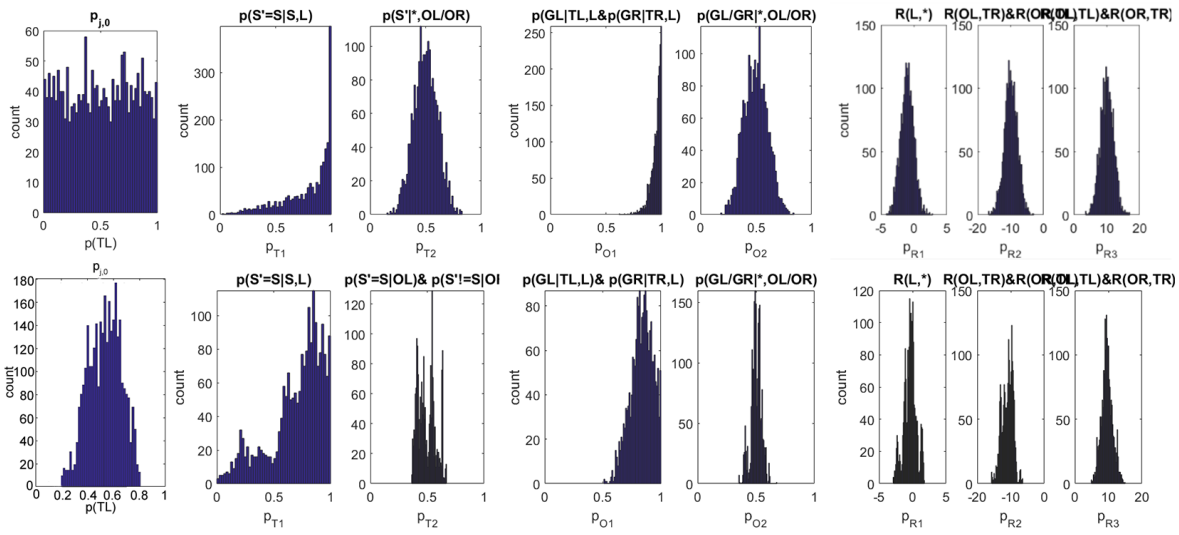


Figure 6: Histograms of assigned priors (top) and learned posteriors (bottom row) over parameters of the agent model in Figure 4 (b): $\theta_{j_2} = \langle$ 0.5, 1, 0.5, 0.95, 0.5, -1, -10, 10 $\rangle$.

Then we use actual observations in the second experiment and run the tiger problem simulator for 30 time steps until the samples converge. In this experiment, we try to learn the model of a classic POMDP agent whose listening capability is as high as 0.95 and the resultant penalty is increased to -10. Namely, as shown in Figure 4 (b), agent j tends to alternately listen and open doors as listening is more expensive now. The actual model of j is $\theta_{j_2} = \langle 0.5, 1, 0.5, 0.95, 0.5, -1, -10, 10 \rangle$ and actual observation sequence consists of {GL,S GR,S GR,S GR,CR GL,S GR,S GR,S GL,CL GR,S GL,CL GR,S GR,CR GR,S GR,CR GL,S GR,S GL,S GR,CL GL,S GL,CL GR,S GR,CR GL,S GR,S GR,S GR,CR GR,S GR,CL GR,S GL,CR}.

The priors we assign to each parameters are shown in the top row of Figure 6. Specifically, they are: $b_j^0 \in U(0,1)$, $p_{T1} \in Beta(2, 0.5)$, $p_{T2} \in Beta(20, 20)$, $p_{O1} \in Beta(19, 1)$, $p_{O2} \in Beta(20, 20)$, $p_{R1} \in N(-1, 1)$, $p_{R2} \in N(-10, 2)$, and $p_{R3} \in N(10, 2)$. We report the learned posterior distributions over model parameters in the bottom row of Figure 6. We can see that the learned parameters are statistically close to the true values.

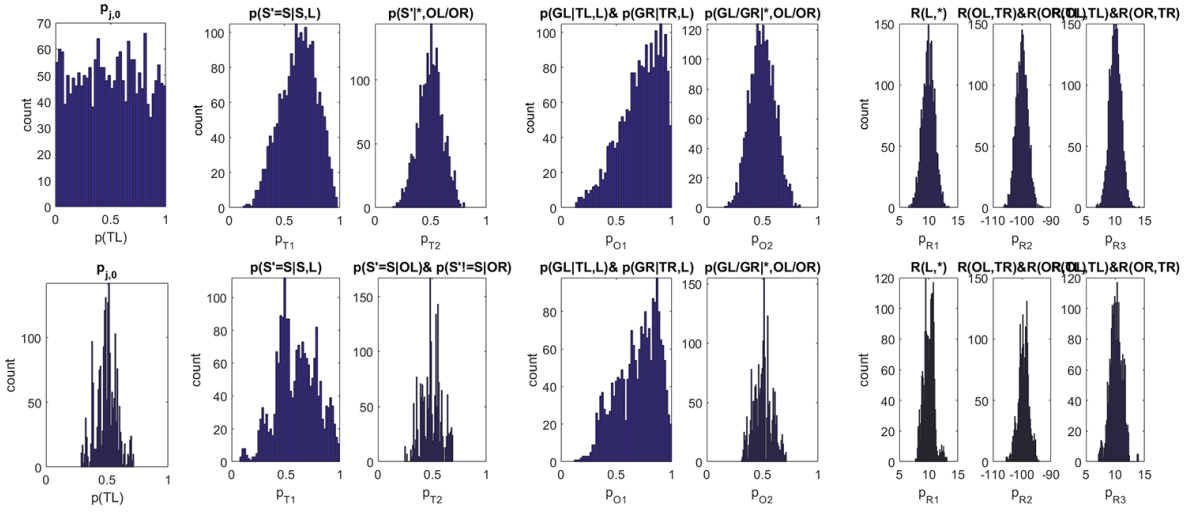


Figure 7: Histograms of assigned priors (top) and learned posteriors (bottom) over model parameters$\theta_{j_3} = \langle 0.5, 0.66, 0.5, 0.85, 0.5, 10, -100, 10 \rangle$ in Figure 4 (c).

In the last experiment, we wants to learn a model of $\theta_{j_3} = \langle$0.5, 0.66, 0.5, 0.85, 0.5, 10, -100, 10$\rangle$. We see that the reward value now equals the listening penalty, therefore the agent always prefers

to listen. We show the marginal posterior distributions over model parameters in figure Figure 7. The experiment was run for only 20 steps since due to the simplicity of the model, and agent is learns from the actual observation sequence of {GL,S GL,S GR,S GL,S GL,CL GR,S GR,S GL,CL GR,S GL,S GL,S GR,S GL,S GL,S GL,S GL,CL GR,S GL,S GL,S GL,S}.

The priors assigned to each parameters are shown in the top row of Figure 7. Specifically, they are: $b_j^0 \in U(0,1)$, $p_{T1} \in Beta(5,3)$ with mode 0.67, $p_{T2} \in Beta(2,2)$, $p_{O1} \in Beta(3.5,1.4)$ with mode 0.85, $p_{O2} \in Beta(2,2)$, $p_{R1} \in N(-1,2)$, $p_{R2} \in N(-100,4)$, and $p_{R3} \in N(10,2)$. We see that while all three parameters of the reward function are learned accurately, the samples of transition and observation function parameters (i.e. p_{T1}, p_{T2}, p_{O1} and p_{O2}) are loosely distributed and not focused tightly on their true values. Intuitively these parameters become less important and can spread widely due to the increased positive reward of listening, $p_{R1} = 10$.

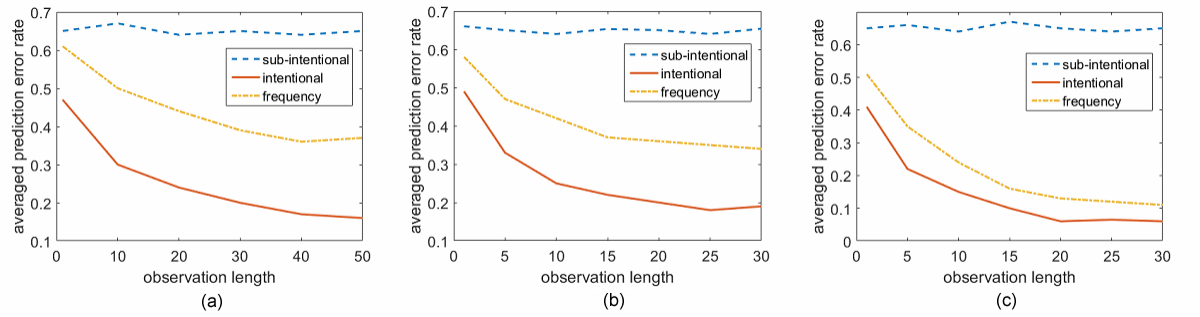


Figure 8: Performance comparisons in terms of prediction error rate vs observation length for $\theta_{j_1} = \langle 0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10\rangle$, (b) $\theta_{j_2} = \langle 0.5, 1, 0.5, 0.95, 0.5, -1, -10, 10\rangle$, and (c) $\theta_{j_3} = \langle 0.5, 0.66, 0.5, 0.85, 0.5, 10, -100, 10\rangle$, as shown in Figure 4.

Because agent i is now able to learn j's likely models, he should be capable of predicting j's actions relatively accurately. Therefore, we tested the performance of our algorithm in terms of prediction accuracy towards j's actions. For conciseness, we show the average prediction error rates for all three experiments in Figure 8. We compare the performance of the intentional model with other subintentional models. The first subintentional model is a frequency-based (fictitious play) model which assumes that agent j's actions are drawn from a fixed but unknown distribution, while the second one is a no-information model which treats actions of agent j as uniform noises. We averaged the experimental results over 10 random runs with 50, 30 and 30 time steps each. Figure 8 shows that, as agent i receives more observations, the intentional I-POMDP approach has the lowest prediction error rate. The fictitious play model is capable of learning and predicting the other agent's actions, but it is not sophisticated enough for such a rational agent j, which reflects on the plot as a higher error rate than the intentional I-POMDP. Lastly, the no-information model has a fixed high error rate as it assumes actions of agent j are drawn from a uniform distribution.

5.1.2 Demonstration of Learning Process in Two-Agent Tiger Problem

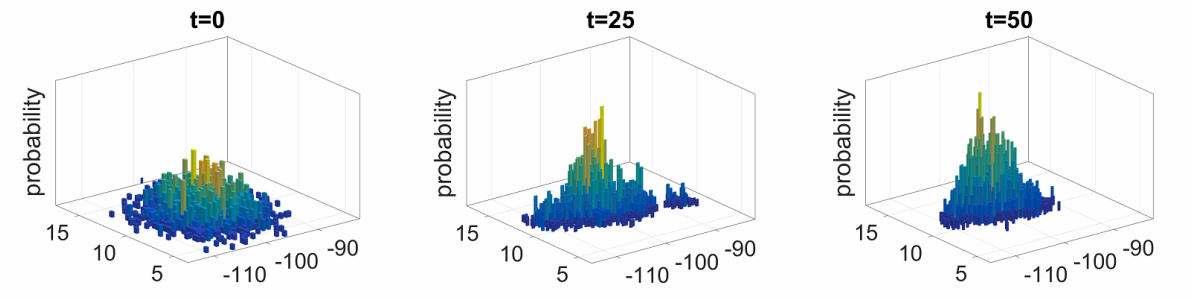


Figure 9: Histogram of all model samples during learning, after projection from 8D to 2D.

In this section, we show a brief demonstration of learning the first model of agent j, θ_{j_1}. Since the original parameter space is 8-dimensional, we use the principal component analysis (PCA) [1] to reduce it to 2D and plot it out as a 3D histogram, as shown in Figure 9. For the illustrative purpose, this time it utilizes a slightly informative prior distribution and gradually converges to the true model. After 50 times steps, the mean value of the sample cluster in Figure 9, $\tilde{\theta}_{j_1} = \langle 0.49, 0.69, 0.49, 0.82, 0.51, -0.95, -99.23, 10.09 \rangle$, is close to the actual model, $\theta_{j_1} = \langle 0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10 \rangle$.

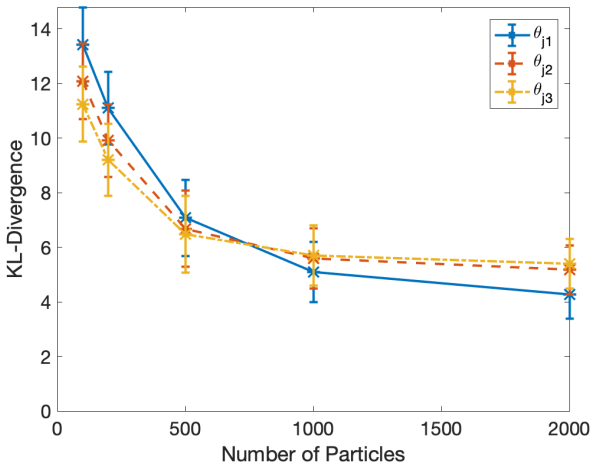


Figure 10: Learning quality measured by KL-divergence improves as the number of particles increases. It measures the difference between the ground truth and the learned model parameters. The vertical bars are the standard deviations. Fixed number of bins (50) are used to compute the discrete probabilities.

In Figure 10 we show that the learning quality of these three experiments in terms of the KL-divergence. It measures the difference between the ground truth and the learned model parameters by giving the relative entropy of the truth with respect to the sampled posterior of models. We define the KL-divergence as the sum of independent KL-divergence of each model parameter. Specifically, $D_{kl}[b(\theta)|\tilde{b}(\theta)] = \sum_{d=1}^{D} D_{kl}[b(\theta_d)||\tilde{b}(\theta_d)]$, where $b(\theta)$ denotes the truth and $\tilde{b}(\theta)$ denotes the sampled posterior, θ is the model represented by 8D parameters, and $d = 1{:}D$ is the dimension. For each parameter dimension d, the KL-divergence reduces to $D_{kl}[b(\theta_d)||\tilde{b}(\theta_d)] = -\log[\tilde{b}(\theta_d \in R)]$ due to $b(\theta_d) = 1.0$ when $\theta_d \in R$, where R is the region / bin containing the ground truth.

5.2 Experiments

In this section, we further experiment with deeper nesting levels in order to analyze the solution quality, convergence and computational cost. Besides the tiger problem, we also experiment with the UAV problem in 2D domain.

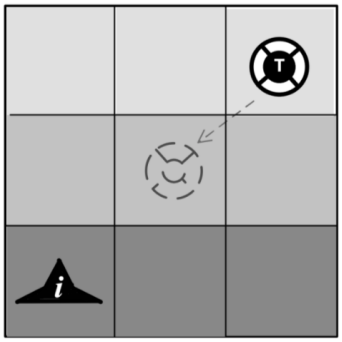


Figure 11: The 3×3 UAV problem.

The multi-agent UAV reconnaissance problem is originally described in (8). Agent j tries to reach a safe house with reward 1, while i tries to intercept j with reward 1 too. Agents have five different actions: move up, move down, move left, move right, and stay. Each move has a cost of 0.04 and is deterministic for both agents. Both agents have sensory accuracy of 0.8 as they are adjacent to each other. This is a larger problem with 36 states, 5 actions, and 3 observations.

5.2.1 Performance Comparison

We firstly fix the modeled agent j to be a level-2 I-POMDP agent and experiment with different modeling approaches for the modeling agent i in order to compare the performance in terms of average reward. We compare level-3, level-2, level-1 intentional I-POMDP models with a subintentional model. In the subintentional model, it is assumed that agent j chooses actions based on a fixed but unknown distribution and therefore is called a frequency-based (fictitious play) model (39).

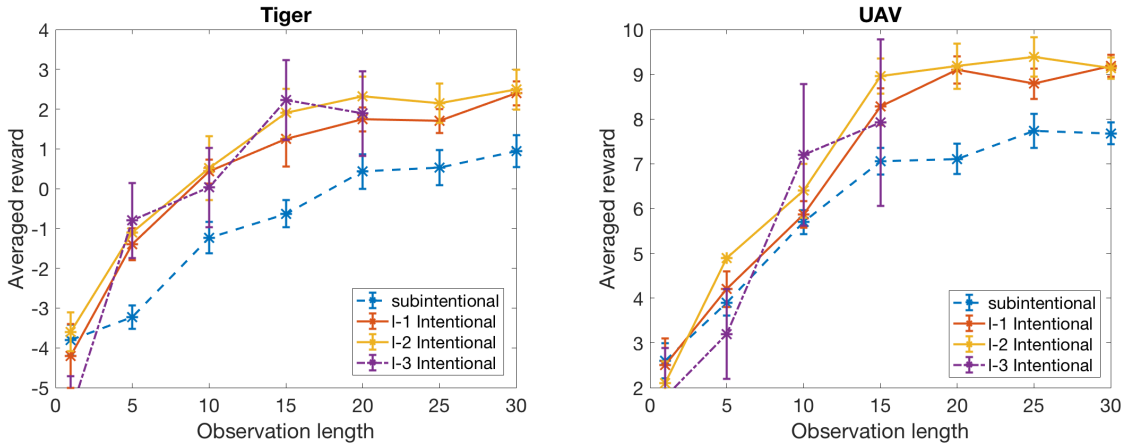


Figure 12: Performance comparison in terms of average reward per time step versus observation length. The plot is averaged on 5 runs and uses 2000 and 1000 samples for Tiger and UAV respectively. The vertical bars stand for standard deviations.

In Figure 12, we see that the intentional I-POMDP approaches has significantly higher rewards as agent i perceives more observations, and level-2 I-POMDP performs slightly better than level-1 while level-3 has high variance but at least competes with level-2. The subintentional approach has certain learning ability but is not sophisticated enough to model a rational (level-2 intentional I-POMDP) agent, therefore its performance is worse than all I-POMDP models. It's worth noting that when the nesting level increases, the performance gain is becoming marginal and theoretically will be bounded by the policy that is generated by an infinite-level nesting model.

5.2.2 Learning Quality Analysis

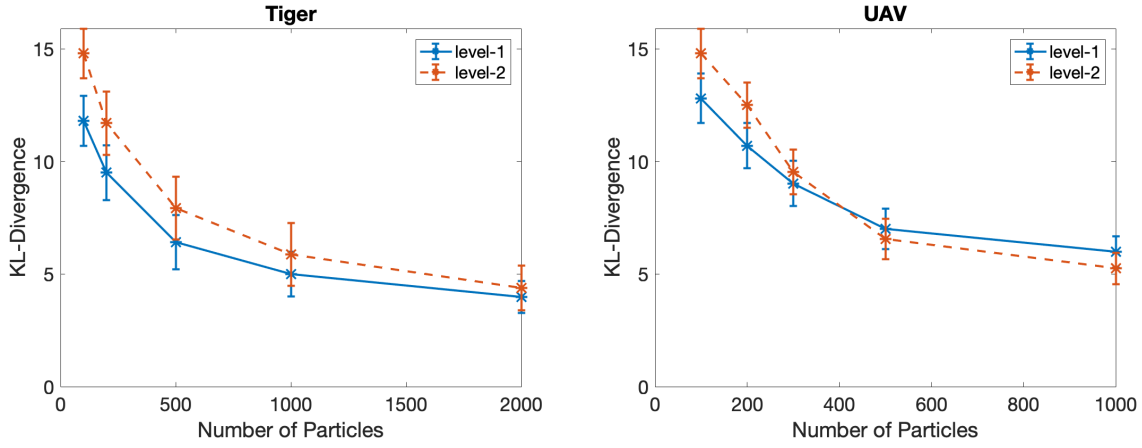


Figure 13: Learning quality, measured by KL-Divergence, improves as the number of particles increases. It measures the difference between the ground truth and the learned model parameters. The vertical bars are the standard deviations.

In Figure 13, we show that the learning quality in terms of KL-Divergence becomes better as the number of particles increases. Although initially level-2 is higher than level-1 due to high dimensional space, the difference quickly becomes smaller in both problems. We see that the KL-divergence of higher-level belief samples are larger in the beginning, but they decrease faster when the number of particles used increases.

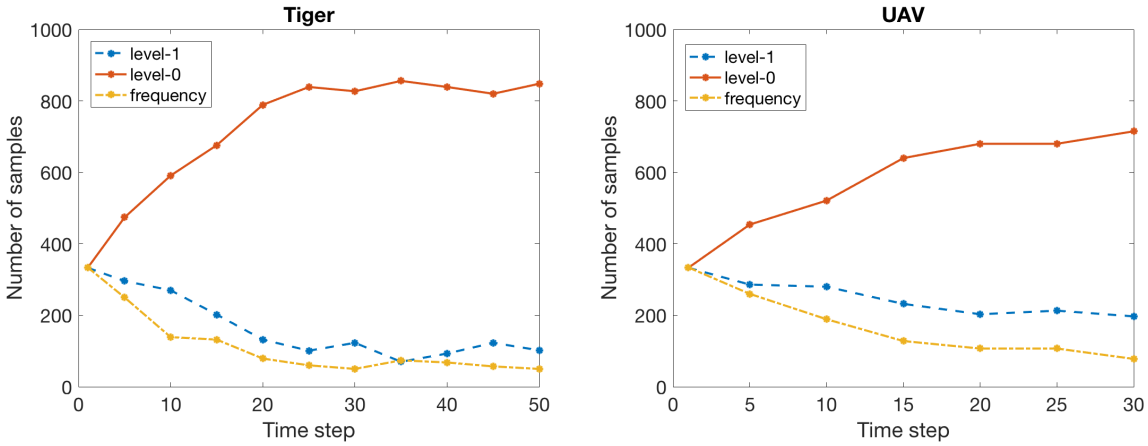


Figure 14: Number of samples representing different nesting levels of j changes as time goes. 1000 samples are used and it starts from equal number (333) of level-1, level-0 and frequency-based samples.

Then we fix the modeling agent i's strategy level to be 2 and try to observe the changes of j's samples which represent different possible models or strategy levels. As shown in Figure 14, we start from equal number of samples that representing j as level-1 I-POMDP, level-0 POMDP, and frequency

based agents, and then gradually learn that the majority of samples converge or become close to the ground truth: j is a level-1 I-POMDP. Notice that the belief samples of lower level do not vanish after the convergence, since the algorithm is unable to completely rule out the possibility of these sample beliefs due to the uncertainty in both agents' models.

5.2.3 Computational Cost

We report the running time of our sampling algorithm in Table IV. The results are averaged over 10 random runs, and the computing machine has an Intel Core i5 2GHz, 8GB RAM, and runs macOS 10.13 and MATLAB R2017.

TABLE IV: Running time for Tiger and UAV problems using various number of samples

Belief Level	N=500	N=1000	N=2000
1	1.96s ±0.43s	3.68s ±1.01s	35.2s ±2.82s
2	5m27.23s ±5.19s	16m36.07s ±10.84s	49m36.07s (single run)

Tiger

Belief Level	N=100	N=500	N=1000
1	4.86s ±1.34ss	12.31s ±1.39s	2m1.43s ±3.29s
2	2m43.1s ±3.98s	9m53.7s ±6.48s	36m19.5s ±18.63s

UAV

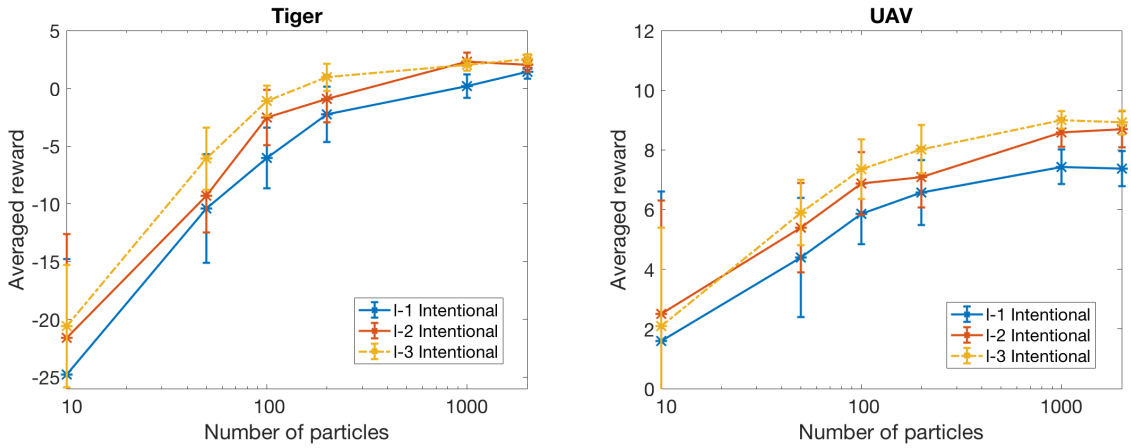


Figure 15: Average reward over number of particles used for nesting level 1, 2 and 3. The x axis is in log scale. The vertical bars are the standard deviations.

Lastly, we analyze the performance gain over computational cost, in terms of average rewards versus number of particles used in the sampling algorithm. In Figure 15, we see that in both tiger and UAV problems, the average reward from using deeper nesting levels are almost higher than that from using shallower nesting levels, which implies the superiority of using deeper nesting levels. However, the overall increment of rewards is diminishing as more particles are used. We think that practically an appropriate balance between actual computational cost and resources needs to be addressed for specific tasks.

CHAPTER 6

A NEURAL NETWORK APPROXIMATION OF I-POMDPS

In this chapter, we will introduce the neural implementation of the I-POMDP framework. The IPOMDP-net is a neural analogy of the I-POMDP framework. It can plan in the same environment as I-POMDP does, where the reward is unobservable and models are known. As a recurrent neural network, it approximates the belief update as well as the policy function that maps the belief states to optimal actions. Similarly to the QMDP-net (10), it combines a parameterized model with an approximate algorithm that solves the model in a single, differentiable neural network. We extend it to the multi-agent setting by embedding the I-POMDP model and the QMDP algorithm into the network architecture. This extension is non-trivial, as will be shown in the following chapters, because embedding an I-POMDP in the network requires encoding the sampling-based belief update algorithm and using sub-network modules to represent the hierarchical interactive belief structure.

Formally, let $I-POMDP_{i,l}(w) = \langle IS_{i,l}(\cdot|w), A, \Omega_i, T_i(\cdot|w), O_i(\cdot|w), R_i(\cdot|w)\rangle$ be the embedded I-POMDP model, where each element is defined the same as in Equation 3.1. Notice that the $I-POMDP_{i,l}(w)$ is now parametrized by w, which are the parameters of the other agent j's model in i's interactive state, $IS_{i,l}(\cdot|w)$, i's own transition function, $T_i(\cdot|w)$, observation function, $O_i(\cdot|w)$, and reward function, $R_i(\cdot|w)$. In the IPOMDP-net approximation, w are also the weights of the neural network. The weights w can be assigned using the true model parameters or learned from random initial values from training. In the first case, the reward is unobservable and models are known therefore is a multi-agent planning problem; in the second case, the reward is usually observable and models are

unknown or assumed unknown, therefore it's a reinforcement learning problem. This chapter focus on the planning problems setting.

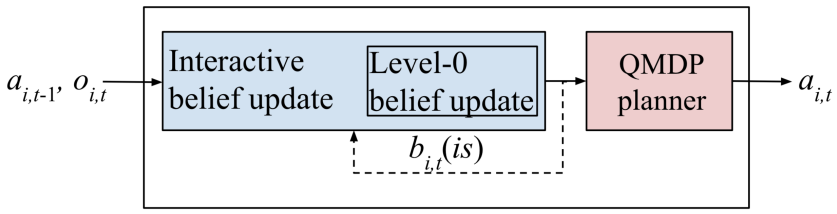


Figure 16: IPOMDP-net architecture overview. It embeds an interactive belief update algorithm and a QMDP planner, the hidden state encodes the interactive belief of agent i.

An IPOMDP-net consists of three main network modules as shown in Figure 16. The first two modules (blue boxes) perform interactive belief update using a customized particle filter for intentional models (53). The interactive belief update module uses the level-0 belief update as a sub-module when the nesting level l bottoms out at 0. The third module (red box) is the QMDP planner, which utilizes the QMDP algorithm to compute the best action given the current belief. The entire network can be trained end-to-end as all modules are differentiable, as shown in Chapter 7. During training, the reward signal $r_{i,t}$ is assumed observable and network weights start from random values.

6.1 Network Architecture

The intuition behind embedding the I-POMDP model and the QMDP algorithm in a single, differentiable neural network is the neural analogy of linear and maximum operations used in the related computations. Namely, the matrix summations and multiplications can be represented by convolutions (convolutional layers) and maximum operations can be represented by max-pooling layers.

Below we will introduce the architectural details of each module with a concrete example of the two-agent tiger problem. For simplicity, let's assume there are two agent i and j in the game and the nesting level is 1. Consider the two-agent tiger game in which two agents are standing in front of two doors. There are a tiger and a pile of gold behind each door. The agents take turns to open doors, they get reward for getting the gold and penalty for facing the tiger. They can choose to hear for further information about the tiger's location, but their hearing is imperfect and they can not directly observe each other's actions.

The IPOMDP-net works on a sampling-based representation of interactive belief state $IS_{i,1} = S \times \theta_{j,0}$. For the tiger game, a sample of physical state can be simply denoted using one-hot vectors, for example [1,0] represents $s = TL$. A sample of j's model can be represented as a vector of length eight, for example $\theta_j = \langle 0.5, 0.67, 0.5, 0.85, 0.5, -1, -100, 10 \rangle$, by parameterizing j's initial belief ($p(TL) = 0.5$), and transition ($p(T_1) = 0.67, p(T_2) = 0.5$), observation ($p(O_1) = 0.85, p(O_2) = 0.5$), and reward functions ($R_1 = -1, R_2 = -100, R_3 = 10$). Thus, an example of initial $IS_{i,1}$ samples can be a 2D vector $\big[[1, 0], [0.5, 1.0, 0.5, 0.85, 0.5, -1, -100, 10]\big]$.

•Interactive belief update module

The core structure of the IPOMDP-net is the interactive belief update module, which is a neural implementation of the sampling based Interactive Belief Update algorithm described in (53). It consists of belief propagation, weighting according to both agents' observations, reweighing and re-sampling. This module embeds both agent i and j's models as network weights w. The output of this module will be the input of QMDP planner module.

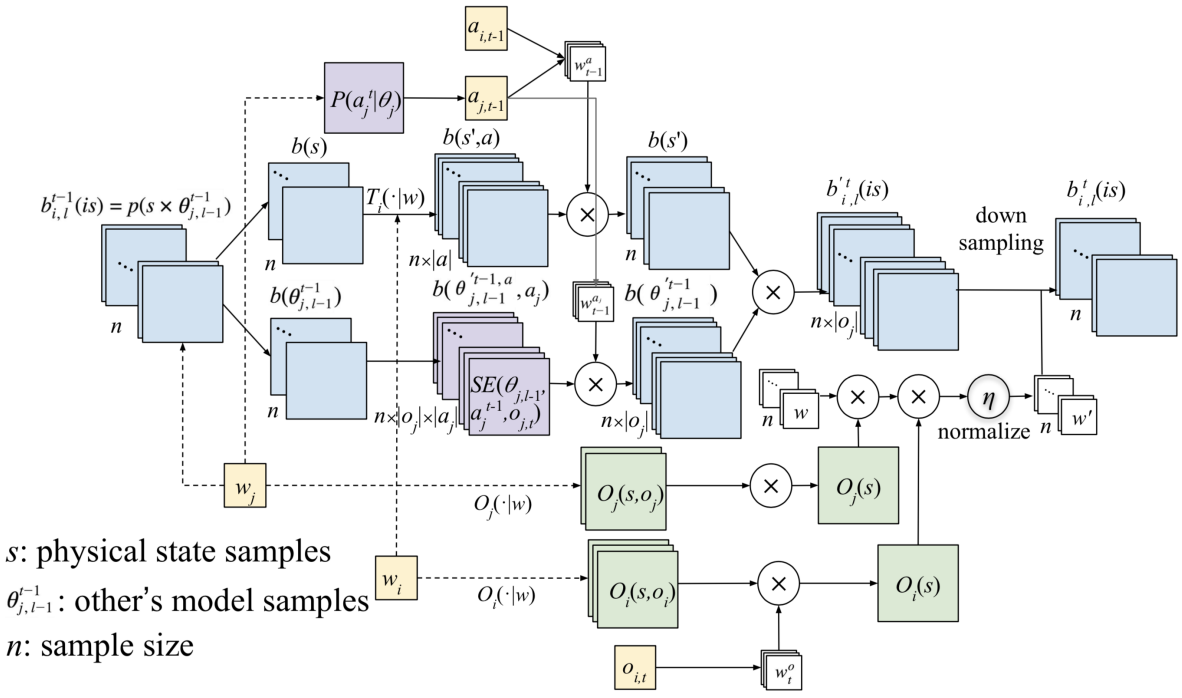


Figure 17: The interactive belief update module

The interactive belief update module maps agent i's interactive belief, action, and observation to a next belief, $b^t_{i,l} = SE(b^{t-1}_{i,l}, a^{t-1}_i, o^t_i)$ (Equation 3.8). It can be divided as two major steps: when agent i performs an action a^{t-1}_i and j performs a^{t-1}_j, i predicts the belief state (Equation 6.1); then when i perceives an observation o^t_i, it corrects and normalizes the prediction (Equation 6.2).

$$\hat{b}^t_{i,l}(is^t) = \sum_{is^{t-1}} b_{i,l}(is^{t-1}) \sum_{a^{t-1}_j} Pr(a^{t-1}_j|\theta^{t-1}_{j,l-1}) T(s^{t-1}, a^{t-1}, s^t) \\ \times \sum_{o^t_j} O_j(s^t, a^{t-1}, o^t_j) \tau(b^{t-1}_{j,l-1}, a^{t-1}_j, o^t_j, b^t_{,l-1}) \tag{6.1}$$

$$b^t_{i,l}(is^t) = \alpha \sum_{a^{t-1}_j} O_i(s^t, a^{t-1}, o^t_i) \hat{b}^t_{i,l}(is^t) \tag{6.2}$$

technically Equation 6.1 is implemented using convolutional layers and sub-modules. Firstly, we get j's all optimal actions according to j's models, $P(a^t_j|\theta^{t-1}_j)$. In Figure 17, as each convolutional channel corresponds to a particular action of j. Notice that the w_i and w_j contain all the parameters of both i and j's models, and $P(a^t_j|\theta^{t-1}_j)$ can be any single-agent POMDP solver, in this case we plug in a pretrained QMDP-net to make the entire network end-to-end. Then agent i's belief, $b(is^{t-1}_{i,1}) = p(s, \theta^{t-1}_{j,0})$, will be divided into s and $\theta^{t-1}_{j,0}$. The first dimension s is convoluted with transition function $T_i(\cdot|w)$ with $|A|$ convolutional filters. The kernel weights are parameters of the transition function $T_i(\cdot|w)$. The output of the convolutional layer is a $D_1 \times D_2 \times n \times |A|$ tensor, where D_1 and D_2 are the sizes of the state space, n is the total number of belief samples, and $|A| = |A_j \times A_j|$ is the number of unique

joint actions of i and j. We stack different input samples together as channels and use conventional 2D convolutions. For instance, for the two-agent tiger game, the predicted physical state after convolution is a $1 \times 2 \times 100 \times 9$ tensor, where the state samples are 1×2 (either $[1, 0]$ or $[0, 1]$), and there are 9 total joint actions ($|L, OL, OR| \times |L, OL, OR|$) and we assume 100 samples are used.

Elements of the second dimension of $is_{i,1}^{t-1}$, the samples of the other agent j's model $\theta_{j,0}^{t-1}$, are the inputs into the level-0 belief update module ($b_{j,0}^{t} = SE(b_{j,0}^{t-1}, a_j^{t-1}, o_j^t)$ in Figure 18. $\theta_{j,0}^{t-1}$ will be updated to a new belief according to the particular model parameters of j. For example, j's model sample [0.5, 1.0, 0.5, 0.85, 0.5, -1, -100, 10] in tiger game will be updated to [0.85, 1.0, 0.5, 0.85, 0.5, -1, -100, 10]. Essentially, only the first parameter will be updated $\big($from $b_j(s{=}TL){=}0.5$ to $b_j(s{=}TL){=}0.85\big)$ as it represents j's belief about tiger being on the left, which corresponds to the $\tau(b_{j,l-1}^{t-1}, a_j^{t-1}, o_j^t, b_{,l-1}^{t})$ function in Equation 6.1. The belief update of j is for any possible action a_j and anticipated observations o_j, so there are totally $n \times |o_j| \times |a_j|$ sub-modules being used.

Since $\hat{b}_{i,1}^{t}(s^t, a)$ after convolution encodes predicted belief about the physical state after taking each of the joint actions, $a \in A = A_i \times A_j$, we want to choose the belief associated with the last joint action. We use a soft indexing similar to the one used in QMDP-net (10), where w_{t-1}^{a} is the indexing vector, a distribution over A. Then we weight $\hat{b}_{i,1}^{t}(s^t, a)$ by w_{t-1}^{a}:

$$\hat{b}_{i,l}^{t}(s^t) = \sum_{a \in A} \hat{b}_{i,l}^{t}(s^t, a) w_{t-1}^{a} \tag{6.3}$$

Similarly, $\hat{b}^t_{i,1}(\theta^{t-1}_{j,0}, a)$ after level-0 belief update encodes predicted models of j after taking each of j's actions, $a \in A_j$, we choose the belief associated with j's last action using soft indexing again. Now $w^{a_j}_{t-1}$ is the indexing vector, a distribution over A_j, and $\hat{b}^t_{i,1}(\theta^t_j, a_j)$ is weighted by $w^{a_j}_{t-1}$:

$$\hat{b}^t_{i,l}(\theta^t_{j,l-1}) = \sum_{a_j \in A_j} \hat{b}^t_{i,l}(\theta^t_{j,l-1}, a_j) w^{a_j}_{t-1} \tag{6.4}$$

After updating j's model samples, we join $\hat{b}^t_{i,1}(s^t)$ and $\hat{b}^t_{i,1}(\theta^t_{j,0})$ together to get the propagated $\hat{b}^t_{i,1}(is^t)$. Namely, the physical state samples $\hat{b}^t_{i,1}(s^t)$ are duplicated by the number of anticipated observations of j and attached with corresponding $\hat{b}^t_{i,1}(\theta^t_{j,0})$. Thus, the number of initial belief samples of i has increased from n to $n \times |o_j|$. The next step is to correct this predicted belief with both agents' observations.

$O_j(s, o_j)$ encodes observation probabilities for each of j's possible observations, it is a $D_1 \times D_2 \times |O_j|$ tensor. Similarly, $O_i(s, o_i)$ encodes observation probabilities for each of i's possible observations, it is a $D_1 \times D_2 \times |O_i|$ tensor. The initial weights of belief samples $w(b^0_i)$ are uniformly initialized as $1/|n|$ and then adjusted by anticipated observations of j and the actual observation of i. We select the observation function corresponding to i's last observation using soft indexing again:

$$O_i(s) = \sum_{a \in A} O_i(s, o_i) w^o_{t-1} \tag{6.5}$$

The remaining step is a simple down-sampling according to the updated weights. After the resampling step, the number of predicted samples of i has reduced from $n \times |o_j|$ back to n.

•Level-0 belief update module

When the nesting level bottoms out, i.e. $l = 0$, this module updates the belief of the agent at level 0, and is used by interactive belief update module at a higher level. For example, in the tiger game, if agent i is at level 1, i models j as a level-0 POMDP and uses this module to compute $b_{i,1}(\theta_{j,0}) = b_{i,1}(b_{j,0}(s), \hat{\theta}_{j,0})$. Thus, j's belief $b_{j,0}(s)$ will be updated in the same way as that in a single-agent POMDP. As shown in Equation 6.6 and Equation 6.7, the two classic steps are the prediction through transition and the correction using observation.

$$\hat{b}_j^t(s^t) = \sum_{s^{t-1} \in S} T(s^{t-1}, a_j^{t-1}, s) b_j^{t-1}(s^{t-1}) \tag{6.6}$$

$$b_j^t(s^t) = \alpha O(s^t, a_j^{t-1}, o_j^t) \hat{b}_j^t(s^t) \tag{6.7}$$

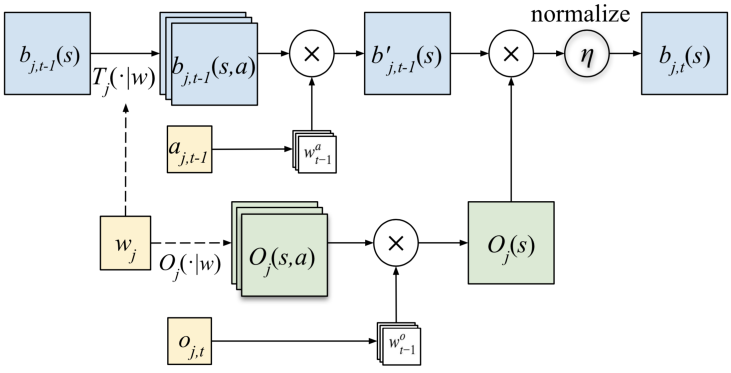


Figure 18: The level-0 belief update module

This module is implemented almost identically to the belief update (filter) module in the QMDP-net (10), except that the transition function T_j and observation function O_j are also input arguments from $b_{i,1}(\theta_{j,0})$. Here we will not repeat the explanation, but the basic idea is to represent the matrix multiplication in Equation 6.6 as a convolutional layer, use soft indexing to select a_j and o_j, and make element-wise multiplication in Equation 6.7.

•QMDP planner module

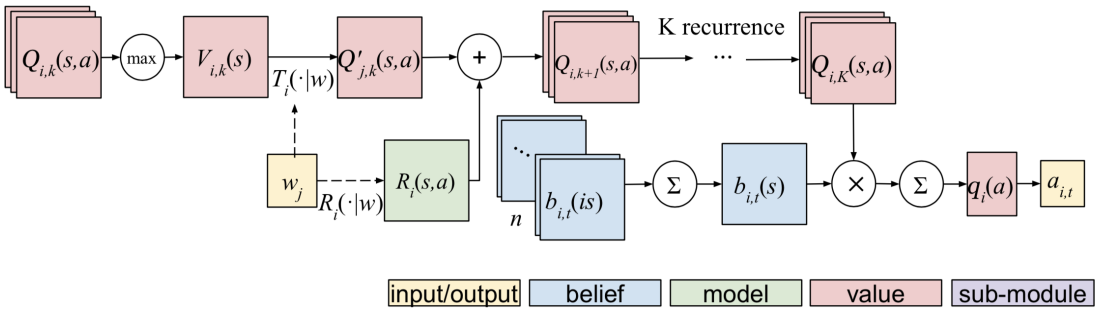


Figure 19: The QMDP planner module

The QMDP planner approximates the I-POMDP value iteration by solving the underlying MDP model, assuming the state is fully observable, and making one-step look-ahead search on the MDP values weighted by i's beliefs. Actions are then chosen according to the weighted Q values. It is similar to QMDP planner in the QMDP-net, except that i's interactive belief (i.e. output of the interactive belief update module) needs to be marginalized over all possible models of j.

$$Q_{i,k+1}(s, a_i) = R_i(s, a_i) + \gamma \sum_{s'} T_i(s, a, s') V_{i,k}(s') \tag{6.8}$$

$$V_{i,k}(s) = \max_{a_i} Q_{i,k}(s, a_i) \tag{6.9}$$

The value iteration in Equation 6.8 and Equation 6.9 is implemented using convolutional and max-pooling layers. The $Q_i(s, a_i)$ is a $D_1 \times D_2 \times |A_i|$ tensor. Equation 6.8 is implemented as a convolutional layer followed by an addition with $R_i(s, a_i)$, the kernel weights encode the transition function T_i. Equation 6.9 is implemented as a max-pooling layer with $Q_{i,k}(s, a_i)$ as input and $V_{i,k}(s)$ as output.

K iterations of value updates are implemented as recurrent layers representing Equation 6.8 and Equation 6.9 K times with tied weights. After K iterations, the approximate Q values for each state-action pair are weighted by i's belief about the physical state (Equation 6.11). But before that, since i's interactive belief contains j's models as well, we need to marginalize over models of j (Equation 6.10). Finally, we select the action that has the highest q-values.

$$b_{i,t}(s) = \sum_{\theta_j} b_{i,t}(is) \tag{6.10}$$

$$Q_i(b_i, a_i) = \sum_{s} Q_{i,K}(s, a_i) b_{i,t}(s) \tag{6.11}$$

6.2 Experimental Results

Since the IPOMDP-net is a neural approximation to the symbolic I-POMDP, we want to know how close this approximation is in terms of planning performance. To make further comparisons, we use known model parameters to initialize the IPOMDP-net weights and substitute QMDP with SARSOP

in the symbolic I-POMDP. Besides the tiger and UAV problems, we also use the maze problem for evaluating the neural I-POMDP planning agent discussed in this chapter and the reinforcement learning agent in Chapter 7.

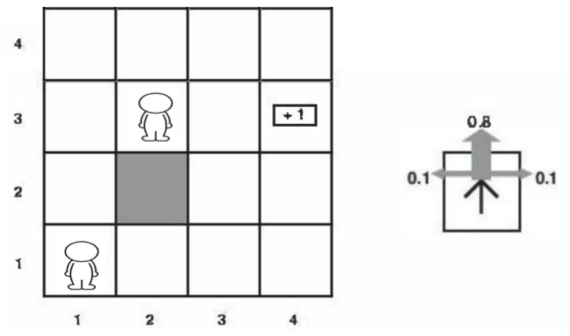


Figure 20: The 4×4 Maze problem. Figure is modified from the similar one in (13).

In a two-agent maze problem, an agent j is tasked with reaching the goal location with reward $+1$. While i also tries to get to the goal, he can get additional $+1$ reward if he catches j. Both agents have imperfect observations with 0.8 accuracy of hearing the other's existence in neighboring locations. Agents have five different actions: move up, move down, move left, move right, and stay. Each move costs them 0.04 and is successful with 0.8 probability except the "stay" action with 1.0.

Three maze sizes are used in the experiments, the 4×4, 10×10 and 16×16. Each of them has 1, 10, and 20 obstacles respectively.

TABLE V: Comparison of average rewards between IPOMDP-net and symbolic I-POMDPs for Tiger and UAV problems. All approaches use QMDP except the one in the last row uses SARSOP.

	Fixed environments	
	Tiger	UAV
IPOMDP-net w/ preassigned weights	2.26 ± 0.16	9.11 ± 0.66
Symbolic I-POMDP	2.27 ± 0.13	9.09 ± 0.75
Symbolic I-POMDP (w/ SARSOP)	2.35 ± 0.20	9.25 ± 0.65

In Tiger and UAV problems, as an approximation / implementation of symbolic I-POMDP, the IPOMPD-net with preassigned weights (1st row) performs almost identically as the corresponding symbolic I-POMDP (2nd row). The symbolic I-POMDP with SARSOP solver (last row) serves as an upper bound but the distance with IPOMDP-net is marginal.

TABLE VI: Comparison of average rewards between IPOMDP-net and symbolic I-POMDPs for different Maze variants. All approaches use QMDP except the one in the last row uses SARSOP.

	Random environments - Maze		
	4×4	10×10	16×16
IPOMDP-net w/ preassigned weights	0.15± 0.05	-0.55 ± 0.12	-0.97 ± 0.13
Symbolic I-POMDP	0.16 ± 0.07	-0.56 ± 0.08	-0.95 ± 0.08
Symbolic I-POMDP (w/ SARSOP)	0.19 ± 0.05	-0.49 ± 0.09	-0.86 ± 0.11

In three Maze tasks, the IPOMDP-net with pre-assigned weights (i.e. the true model parameters) performs almost the same as its symbolic I-POMDP counterpart. In the last rows of both tables, we report additional results on the symbolic I-POMDP using the SARSOP planner (51) instead of the QMDP. The SARSOP represents the best performance that a symbolic approach can achieve now. While the symbolic I-POMDP with SARSOP performs the best in almost all tasks, we see the IPOMDP-net (with QMDP) has comparable performance. We will look into ways of implementing SARSOP or other planners in IPOMDP-net, which generally requires more sophisticated network design.

CHAPTER 7

REINFORCEMENT LEARNING USING RECURRENT NEURAL NETWORKS

In this chapter, we apply the same network architecture as in the IPOMDP-net to reinforcement learning problems, in which the reward signals are often observable but the model (transition, observation and reward functions) is unknown or assumed unknown. We want to train a neural agent that can learn a compact representation of the model of the simulated environment and therefore has the capability to generalize its learned policy to larger, unseen domain.

We firstly introduce the training algorithm, then discuss the model-free network for comparison, and show experimental results using Tiger, UAV and Maze problems.

7.1 Training Using Reinforcement Learning

From an agent's perspective, the reward signal from the environment is now observable. We want to train an neural agent that can optimize its randomly initialized weights to minimize a certain loss function so that it can generate optimal or near-optimal actions in such an environment (Figure 21).

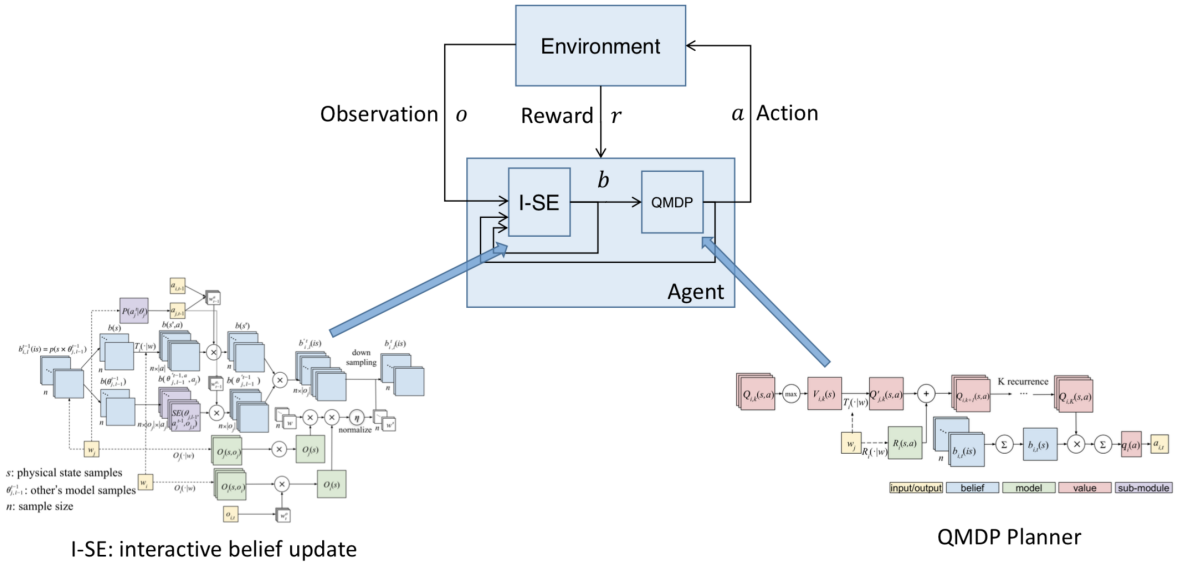


Figure 21: Model-based deep reinforcement learning from an agent's perspective. The rewards are observable, the interactive state update (I-SE) and QMDP modules use the same architectures as in the IPOMDP-net, but with unknown, randomly initialized weights.

Recall that the optimal action-value function satisfies the Bellman equation:

$$Q^*(b,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(b',a')|b,a] \tag{7.1}$$

which motivates us to update the neural network weights by minimizing the difference between the optimal Q-value with the current prediction from the network:

$$\mathcal{L}(w_i) = \mathop{\mathbb{E}}_{b,a\sim p(b,a)}[(r + \gamma \mathop{max}_{a} Q(b', a'|w_{i-1}) - Q(b, a|w_i)]^2 \tag{7.2}$$

Where $p(b, a)$ is the behavior distribution which is followed by the agent to choose its actions according to different beliefs, e.g. a greedy strategy that always selects the action associated with the maximum Q-value. Practically, instead of directly minimizing this loss over all data, stochastic gradient descent (SGD) is often used to minimize it using a single sample or a batch of samples from the training data set.

$$\mathcal{L}(w_i) = [(r + \gamma \mathop{max}_{a} Q(b', a'|w_{i-1}) - Q(b, a|w_i)]^2 \tag{7.3}$$

where $(r + \gamma\, max_a\, Q(b', a'|w_{i-1})$ is usually called the optimizing target, and $Q(b, a|w_i)$ is the network prediction.

$$\nabla\mathcal{L}(w_i) = \partial\mathcal{L}(w_i)/\partial w_i = 2[(r + \gamma \mathop{max}_{a} Q(b', a'|w_{i-1}) - Q(b, a|w_i)]\partial Q(b, a|w_i)/\partial w_i \tag{7.4}$$

This gradient can be computed simply using the partial derivatives with respect to the weights:

$$w_i = w_i + \alpha\nabla\mathcal{L}(w_i) \tag{7.5}$$

We train the IPOMDP-net in a reinforcement learning setting following the similar way in DQN (27) (Algorithm 6). Due to partial observability, we need to use belief state instead of physical state in the experience replay memory $[b_{i,t}, a_{i,t}, r_{i,t+1}, b_{i,t+1}]$. To update the network parameters w and back-propagate the errors, we define the loss function as the mean squared error between the Q-value of the target and the IPOMDP-net. Immediate rewards are assumed observable and agent i's belief is computed in the belief update module.

Algorithm 6: Deep Q-learning with Experience Replay

Initialize belief state, replay memory, nesting level l

Initialize the networks with random weights w

For episode$= 1$ to M:

  Initialize $a_{i,0}$ and get $o_{i,1}$

    for $t = 1$ to T:

      Sample $a_j^t \sim P(A_j|\theta_{j,l-1})$

      Select a random action $a_{i,t}$ with probability ϵ

      Otherwise select $a_{i,t} = \arg\max_a Q(b_{i,t}, a_i; w)$

      Execute action $a_{i,t}$, obtain reward $r_{i,t}$, observation $o_{i,t}$, and updated belief $b_{i,t+1}$

      Store $[b_{i,t}, a_{i,t}, r_{i,t+1}, b_{i,t+1}]$ in replay memory

      Randomly sample a minibatch in replay memory $[b_{i,m}, a_{i,m}, r_{i,m+1}, b_{i,m+1}]$

      Compute the target Q-value:

$$y_m = \begin{cases} r_{i,m} & \text{if m is the terminal step} \\ r_{i,m} + \beta \max_{a_i} \hat{Q}(b_{i,m+1}, a_i'; w^-) & \text{otherwise} \end{cases}$$

      perform gradient descent on: $(y_m - Q(b_{i,m}, a_{i,m}; w))^2$

To successfully optimize the network parameters, there are three major techniques we used to stabilize the learning process. First, we use samples of experience replay tuples $[b, a, r, b']$ instead of sequential data generated by the simulation. These samples from experience replay memory are used in batches when training the neural network, which breaks the sequential dependence in the data and satisfies the i.i.d. assumption in the deep learning methods. Second, we keep a separate target network to provide updates to the main network, meaning that the parameters from the previous iteration, w_{t-1}, are held fixed when optimizing the loss function $\mathcal{L}(w_t)$ for the current iteration. This technique makes labels of data unchanged for a short amount of time, which is similar to supervised learning, and helps stabilize the learning process. Third, we use a particular adaptive optimizer, the RMSProp (50), other than the simple SGD or batchSGD, because it can regulate the parameter learning rate and has been proved very effective in various DRL training practice (27; 10; 31).

Besides, we know that a well known problem with QMDP approximation for POMDP is that MDP policies do not execute actions that reduce the uncertainty of the belief (i.e. no information gathering). The training algorithm overcomes this by using this ϵ-greedy exploration heuristic. Note that in our training algorithm, the ϵ-greedy strategy ensures adequate exploration of the belief space: we select a random action with probability ϵ, or the current best action (that maximizes Q) with probability 1-ϵ. This is known as the exploration-exploitation dilemma which can be solved simply yet very effectively by the ϵ-greedy strategy.

7.2 Model-Free Networks for Comparison

Intuitively, in a multi-agent sequential decision making problem, others' models will cause their actions which indirectly impact our actions. Thus, having a model of the other agent should benefit our

decision making in such environments, because we can learn to plan effectively using the models of the world and others, instead of learning a reactive mapping from observations to actions. Therefore, we are interested in comparing a state-of-the-art model-free network with our IPOMDP-net in terms of policy quality.

In order to compare the model-based IPOMDP-net with other model-free networks, we also modify and implement a model-free network that acts similarly to the action-specific deep recurrent Q-network (ADRQN) for single-agent domain (54).

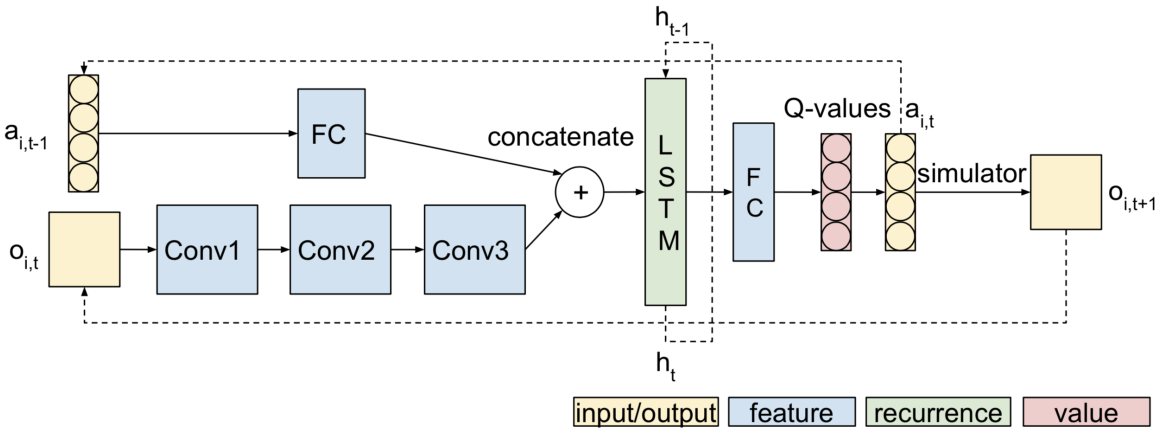


Figure 22: One time slice of ADRQN. Inputs are $a_{i,t-1}$ and $o_{i,t}$, outputs are $a_{i,t}$ and $o_{i,t+1}$. FC stands for fully connected layer. Conv stands for convolutional layers. Output of LSTM layer h_t will be input into LSTM in the next time slice. Actual hyper parameters vary on different problems.

As shown in Figure 22, this network is a dual-modal hybrid architecture that learns from the action and observation histories, i.e. $[a_{i,0}, o_{i,1}], [a_{i,1}, o_{i,2}], ..., [a_{i,t-1}, o_{i,t}]$. The time series of action-observation pairs are integrated by an LSTM layer that extracts features (from these pairs) and learns the latent states. Then a fully connected layer computes Q-values based on the learned latent states. Intuitively, these latent states integrates the information contained in action and observation histories. It has been shown that the ADRQN (54) outperforms the DRQN (31), which outperforms the DQN (27). Thus, it is one of the state-of-the-art model-free networks for sequential decision making.

Training the ADRQN is similar to training the DQN except the experience replay memory now changes from $[s_t, a_{i,t}, r_{i,t}, s_{i,t+1}]$ to $\left[\{a_{i,t-1}, o_{i,t}\}, a_{i,t}, r_{i,t}, o_{i,t+1}\right]$ due to partially observability. Training ADRQN also converts a learning problem to a high-dimensional non-convex function optimization (on the network weight w space). However, besides model-embedding and network architecture, the major difference between IPOMDP-net and ADRQUN is that weights in the IPOMDP-net encode both agents' model parameters, but in ADRQN the weights are from another parameter space. Thus, after the training converges, the weights of IPOMDP-net represent approximate model parameters, while the weights of ADRQN seem to be "random" values used to approximate the policy function.

7.3 Experiments

7.3.1 Experimental setup

The goal of the experiments is to verify our assumption that embedding models of other agents in the planning network benefits our decision-making. We want to understand the benefits in terms of the policy quality and generalizability.

There are two phases of experiments: training and testing. The training phase is for different neural networks (IPOMDP-net and ADRQN) to be trained in a reinforcement learning way so that the converged networks are approximations to the true policy function. The testing phase is to evaluate the trained networks in testing environments and compare their performance with the symbolic I-POMDP.

We designed experiments of five problems in two categories. In the first category, we use the two-agent tiger game problem and UAV problem, in which the problem environments are small and fixed. The neural networks are trained on the same, fixed environment, and then applied back to it for testing. In the second category, we use three variations of the Maze problem (55) with size 4×4, 10×10, and 16×16, in which the agent j tries to reach the goal while i tries to reach the goal and / or catch j. We want to evaluate if the model-free networks can keep up with our model-based network and the policy learned in smaller environments (10×10) can generalize to larger ones (16×16). In these Maze variations, the locations of the start, goal and obstacles are random in each of the training and testing maps.

For the Maze problem, we train the IPOMDP-net on 4×4 and 10×10 variations, and directly increase the recurrence K of the trained network in 10×10 settings to 40 and test it in 16×16 maps. There are 2, 10, and 20 obstacles in each Maze variation. The locations of the start, goal and obstacles are random in each of the training and testing maps. We train the networks on randomly generated environments and test them on new, unseen ones. In all the experiments, agent i does not have access to the ground truth, i tries to learn both agents' models through playing in the simulator, assuming reward signals are obtainable.

7.3.2 Results and Discussions

In Table VII and Table VIII, we report the average rewards in Tiger, UAV, and three Maze variations. We train the IPOMDP-net starting from random weights in a RL setting.

TABLE VII: Comparison of average rewards between the model-based IPOMDP-net and ADRQN for Tiger and UAV problems.

	Fixed environments	
	Tiger	UAV
ADRQN	2.29 ± 0.35	8.98 ± 0.97
IPOMDP-net w/ trained weights	2.32 ± 0.29	9.19 ± 0.72

In Tiger and UAV problems, the learning and testing environments are fixed. We see that in such environments, the performance difference is small. The model-based IPOMDP-net outperforms the model-free ADRQN in the UAV problem.

TABLE VIII: Comparison of average rewards between the model-based IPOMDP-net and ADRQN for different Maze variants.

	Random environments - Maze		
	4×4	10×10	16×16
ADRQN	0.12 $\pm$ 0.08	-0.73 $\pm$ 0.17	-1.58 $\pm$ 0.39
IPOMDP-net w/ trained weights	0.18 $\pm$ 0.05	-0.52 $\pm$ 0.10	-0.88 $\pm$ 0.21

In three Maze tasks, the learning maps are randomly generated and testing maps are new unseen ones. The performance of the model-free ADRQN degrades faster as maps size increases.

•IPOMDP-net learns policies that generalize to new environments.

For the 4×4 Maze problem, we randomly generate 1100 maps and divide them into training and testing sets of size 1000 and 100. For the 10×10 Maze, there are 5000 maps for training and 200 for testing. We see that in the first and second columns of Table VII, the average rewards of IPOMDP-nets (both trained and pre-initialized) are higher than the ADRQN.

In the fixed Tiger and UAV environments (Table VII), the model-free ADRQN has comparable performance to the trained IPOMDP-net. The reason is that in a fixed environment, a network may directly learn the mapping from features to policy. On the contrary, the trained IPOMDP-net computes a near-optimal policy for any given environment, therefore it essentially learns a model for planning.

•IPOMDP-net policy learned in small environments transfers directly to larger ones.

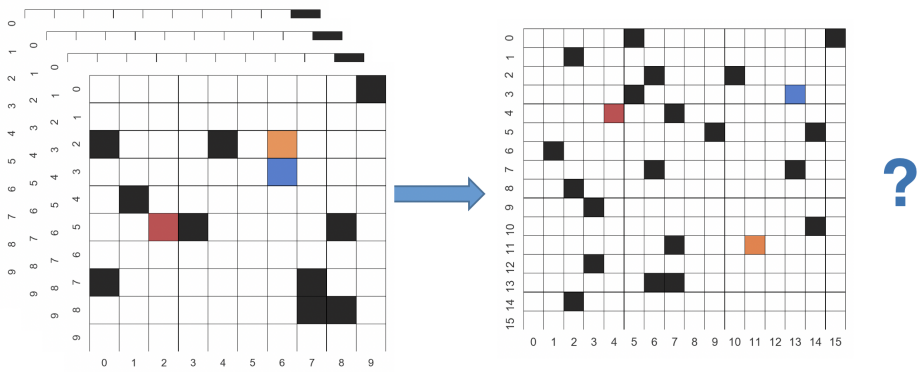


Figure 23: Network is trained in 10×10 random maps and tested in larger, unseen ones in size 16×16. Black denotes obstacles, orange is j's location, red is i's location, and blue is the goal location.

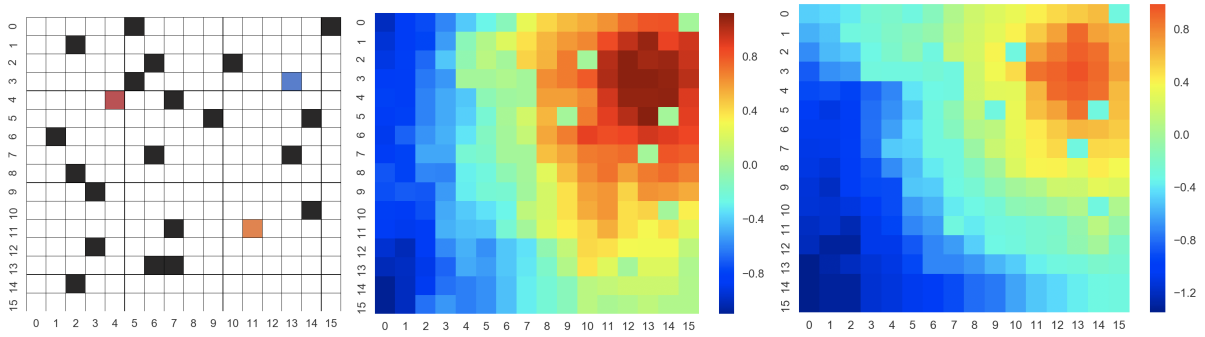
For the 16×16 Maze problem, we directly apply the IPOMDP-net trained in 10×10 mazes and increase the value iteration recurrence K to 40, keeping all other parameters unchanged. For the trained ADRQNs in 10×10 mazes, we also increased the recurrence of LSTM layers to 20. Although the map size is larger, the underlying planning nature is the same as in smaller mazes. From the last column of Table VIII, we see that the IPOMDP-net is significantly better than the model-free ADRQN. When the maze size increases (Table VIII row 1 from left to right), the performance of ADRQN degrades fast. Since embedding the models and planning algorithm in the network enables the agent to learn to plan, we think that as long as the environment to transfer has the same or similar planning nature, a similar performance as in the maze problem should be expected.

•IPOMDP-net learns an overall better policy instead of a "true" model.

If we compare the performance of pre-initialized IPOMDP-net in Table VI of Chapter 6 and the trained one in Table VIII, the trained IPOMDP-net performs better than the pre-initialized one. Intu-

itively the learning algorithm is optimizing over a large parameter space. It makes sense that the learned model should be the ground truth if the embedded I-POPMDP algorithm is exact. Since our the planning algorithm is QMDP, the learned $T(\cdot|\theta)$, $O(\cdot|\theta)$, and $R(\cdot|\theta)$ does not necessarily represent the true transition, observation, and reward functions. The end-to-end training enables the IPOMDP-net to learn an useful model, instead of a "correct" one, that compensates the limitation of the QMDP planner. Our result coincides with the one in the single-agent domain (10).

7.4 Visualization



(a) (b) (c)

Figure 24: Visualization of both agents' value functions on Maze 16×16 problem: (a) a particular game map, (b) learned value function of i on one sample state (when j at the orange square position), (c) learned value function of j for the 16×16 Maze problem. Black squares are obstacles, red is i's location, orange is j's location, and blue is target location.

We visualize the learned value function of agent i for the 16×16 Maze problem. The particular map setting is shown in Figure 24(a), where black squares represent obstacles, red represent i's location, orange represent j's location, and blue is the goal. In Figure 24(b), we see agent i assigns high values over the target and j's locations, but the target location is "hotter"than j's location. This is because that j tends to move a lot, and therefore, its location is not as valuable as the goal (catching j or reaching target location gives the same $+1$ reward). In Figure 24(c), since j is modeled as a level-0 POMDP agent, j has no clue about i's existence. Thus, in j's reward function, states close to the goal have high values.

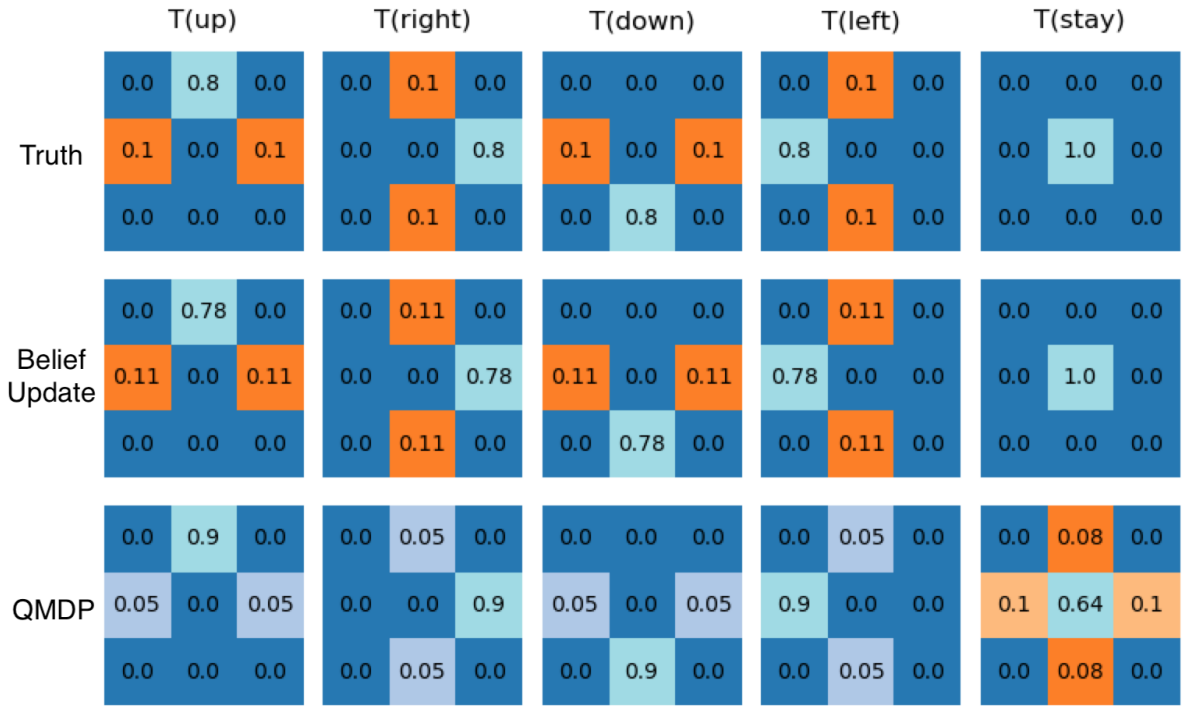


Figure 25: The learned transitions in belief update module and QMDP module are different. The first row is the ground truth. The second row is the transition in belief update module. The third row is the transition in QMDP module.

As an example of learned "incorrect" models, we also visualize the learned transition functions in the interactive belief update and the QMDP planning modules. We see that in Figure 25, the transition learned in the interactive belief update module is close to the truth, while the transition in QMDP is quite different. The different weights learned from training allow each module to choose its own approximation to mitigate limitations of the QMDP approximation.

7.5 Technical Details

In our experiments, we represent $T_i(\cdot|w), O_i(\cdot|w), R_i(\cdot|w)$ with various CNN kernels depending on specific problems, since each problem has unique transition, observation and reward functions. We restrict ourself to use softmax activations for the $T_i(\cdot|w)$ and sigmoid activations for the $O_i(\cdot|w)$ on the convolutional kernels, since $T_i(\cdot|w)$ and $O_i(\cdot|w)$ are both probability distributions.

For the IPOMDP-net, we use the RMSProp optimizer (50) and 1×10^{-3} learning rate. We set the minibatch size to be 32 and the replay memory size to be $100,000$. We use 1.0 as the initial exploration rate ϵ, and 1×10^{-3} as the decay rate. Other parameters vary on specific tasks.

Belief Update Convergence

Using the tiger game as an example, we plot out the KL-divergence between the ground truth and learned model parameters (Figure Figure 26). It measures the quality of the interactive belief update module in the IPOMDP-net, and therefore, provides an error bound on the learned interactive belief samples. We see that the belief estimate is converging to the true value with the increasing number of samples, which provides accurate belief "features" for the QMDP module to compute corresponding policies.

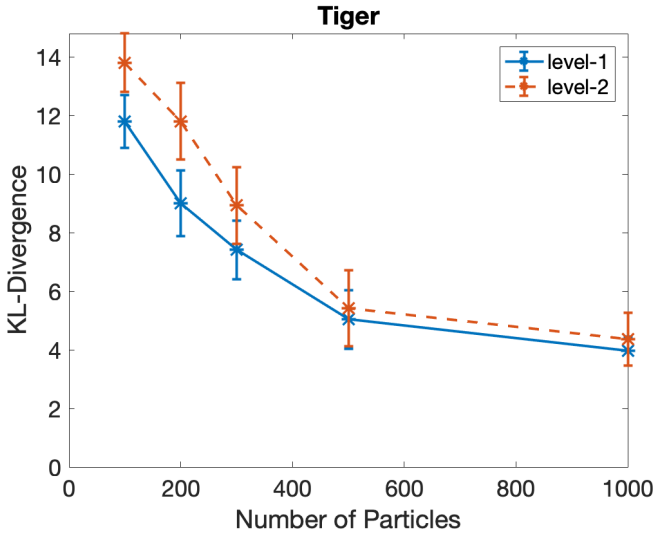


Figure 26: The KL-Divergence decreases as the number of samples increases on nesting level 1 and 2. It measures the difference between the ground truth and the learned model parameters. The vertical bars are the standard deviations.

Training Evaluation

We plot the performance as a function of training length in Figure 27 using the tiger problem. We observe an interesting pattern that the model-based IPOMDP-net converges faster than the model-free ADRQN, the latter appears to take some additional time to enter the decent performance region. We think that it is due to the implicit representation of "belief" (i.e. the hidden layer after the LSTM) in the ADRQN, which requires more training data to converge to the truth. We also draw the horizontal line of symbolic I-POMDP for a comparison.

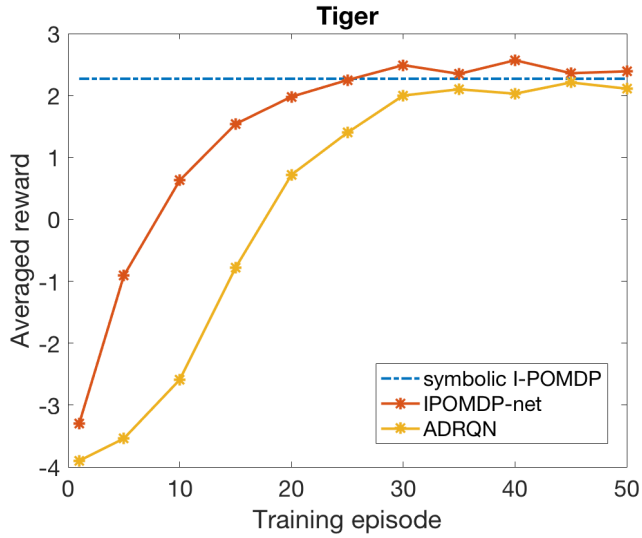


Figure 27: Performance on tiger game versus training episodes. The horizontal line is the symbolic I-POMDP for reference.

Runtime Comparison

We show the runtime comparison between the symbolic I-POMDP and the trained neural IPOMDP-net in Figure 28. We train and test the Tensorflow implemented IPOMDP-net on an AWS g.2 instance with a NVIDIA GRID K520 GPU, which was launched in 2013. We test the symbolic I-POMDP on a Intel Core i5-6360U CPU, which was launched in 2015.

Compared with the symbolic approach, since the IPOMDP-net did not further address the curse of dimensionality in belief update or the curse of history in planning, the runtime of IPOMDP-net is expected to grow exponentially as the reward increases (which needs more samples and deeper nesting

levels (53)). However, due to the efficient computation of matrix operations on GPU versus CPU, this engineering optimization buys us more space in which we hope to fit in more realistic problems from the multi-agent domain.

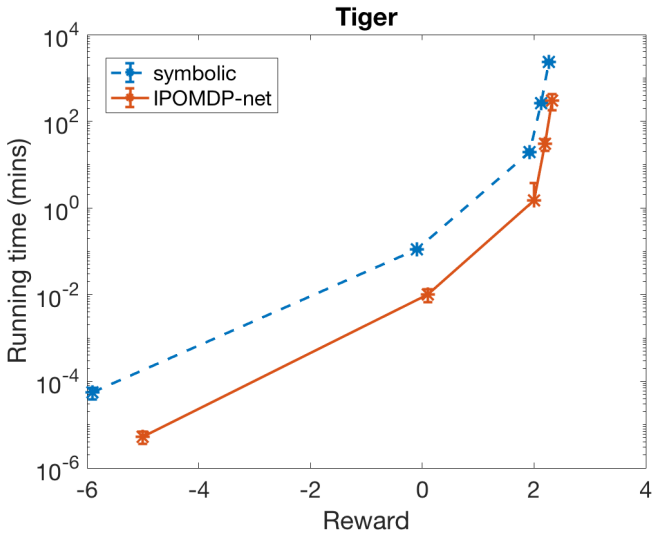


Figure 28: Runtime verse increasing rewards in the tiger game. Although runtime in both approaches seems to be exponentially increasing, the symbolic I-POMDP is slower at similar reward point for roughly an order of magnitude.

CHAPTER 8

CONCLUSION AND FUTURE DIRECTIONS

We have described a novel approach to learn other agents' intentional models in a multi-agent, partially observable, stochastic environment. We approximate the exact interactive belief update using Bayesian inference and customized sequential Monte Carlo methods. We show the correctness of our theoretical framework using the multi-agent tiger and UAV problems, in which the algorithm accurately learns others' models over the entire intentional model space. We give comparisons of policy quality and analysis of the performance of deeper strategy level. Overall, our approach provides a generalized Bayesian learning method to learn other agents' belief, strategy level, and transition, observation and reward functions.

We have described the IPOMDP-net, a neural network architecture for multi-agent sequential decision making. It embeds an I-POMDP model and a QMDP planning algorithm in a network architecture and can be used as a neural agent in a planning setting. We show similar performance between the symbolic and neural I-POMDP approaches while the latter computes faster than the former due to the computation happens on dedicated, more efficient neural computing devices.

We have also applied the IPOMDP-net architecture in reinforcement learning problems in which the rewards are known. Through training by deep reinforcement learning, the randomly initialized weights of the NN are optimized to maximize the quality of the agent's policy. We show the effectiveness, performance, and generalizability of trained agents on various problems. Our model-based neural agent

which learns to plan outperforms the state-of-the-art model-free network which only learns reactive policies.

We provide potential future research directions as follows:

For the symbolic approach, more efforts can be made on leveraging nonparametric Bayesian methods which inherently deal with nested belief structures. Intuitively, since the value function of I-POMDP over the continuous belief space remains piece-wise linear convex, there are finite number of interactive belief "regions" which correspond to different optimal actions. At each nesting levels, such one-to-one correspondence remains. This motivates us to associate the nested belief structure in intentional models with commonly used stochastic processes in Bayesian nonparametric, such as the Chinese restaurant process (CRP) (56). Besides, there have been recent advances in applying nonparametric Bayesian methods on subintentional I-POMDP models (57), which implies the feasibility of similar applications on equivalent intentional models. Lastly, the experiential problems presented in this paper can be extended to more complicated multi-agent tasks.

For the neural approach, one future direction is to implement the exact I-POMDP value iteration or other more powerful planners instead of the QMDP approximation. The first step shall be an implementation of single-agent POMDP value iteration, which is straightforward as it only involves linear matrix operations and maximum operations. The real challenge is to find the most appropriate way to represent the value function on the nested interactive belief structures. This may lead to another direction, which is a better, concise representation of the interactive belief in multi-agent problems under the neural analogy. In this paper, we mitigate the curse of dimensionality from the belief state space by encoding a sampling algorithm in the IPOMDP-net and operating on the sample space instead. We might be

able to use feature embeddings to create reduced-dimension vectors or matrices to effectively represent interactive beliefs as concise and unique features that still contain the valuable nested information.

APPENDIX

COPYRIGHTS

CITED LITERATURE

1. Kaelbling, L. P., Littman, M. L., and Cassandra, A. R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1):99–134, 1998.

2. McCallum, A. K. and Ballard, D.: Reinforcement learning with selective perception and hidden state. Doctoral dissertation, University of Rochester. Dept. of Computer Science, 1996.

3. Liu, M., Liao, X., and Carin, L.: The infinite regionalized policy representation. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pages 769–776, 2011.

4. Doshi-Velez, F., Pfau, D., Wood, F., and Roy, N.: Bayesian nonparametric methods for partially-observable reinforcement learning. IEEE transactions on pattern analysis and machine intelligence, 37(2):394–407, 2015.

5. Gmytrasiewicz, P. J. and Doshi, P.: A framework for sequential planning in multi-agent settings. J. Artif. Intell. Res.(JAIR), 24:49–79, 2005.

6. Panella, A. and Gmytrasiewicz, P. J.: Bayesian learning of other agents' finite controllers for interactive POMDPs. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pages 2530–2536, 2016.

7. Papadimitriou, C. H. and Tsitsiklis, J. N.: The complexity of Markov decision processes. Mathematics of operations research, 12(3):441–450, 1987.

8. Doshi, P. and Gmytrasiewicz, P. J.: Monte Carlo sampling methods for approximating interactive POMDPs. Journal of Artificial Intelligence Research, 34:297–337, 2009.

9. Douced, A., de Freitas, N., and Gordon, N.: An introduction to sequential Monte Carlo methods. Sequential Monte Carlo Methods in Practice, pages 3–13, 2001.

10. Karkus, P., Hsu, D., and Lee, W. S.: QMDP-Net: deep learning for planning under partial observability. In Advances in Neural Information Processing Systems, pages 4697–4707, 2017.

11. Littman, M. L., Cassandra, A. R., and Kaelbling, L. P.: Learning policies for partially observable environments: Scaling up. In Machine Learning Proceedings 1995, pages 362–370. Elsevier, 1995.

12. Marthi, B., Pasula, H., Russell, S., and Peres, Y.: Decayed MCMC iltering. In Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence, pages 319–326. Morgan Kaufmann Publishers Inc., 2002.

13. Russell, S. and Norvig, P.: Artificial intelligence: A modern approach. 2009.

14. Ross, S., Chaib-draa, B., and Pineau, J.: Bayes-adaptive POMDPs. In Advances in neural information processing systems, pages 1225–1232, 2008.

15. Bernstein, D. S., Zilberstein, S., and Immerman, N.: The complexity of decentralized control of Markov decision processes. In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence, pages 32–37. Morgan Kaufmann Publishers Inc., 2000.

16. Seuken, S. and Zilberstein, S.: Improved memory-bounded dynamic programming for decentralized POMDPs. arXiv preprint arXiv:1206.5295, 2012.

17. Ng, B., Boakye, K., Meyers, C., and Wang, A.: Bayes-adaptive interactive POMDPs. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012.

18. Rosman, B., Hawasly, M., and Ramamoorthy, S.: Bayesian policy reuse. Machine Learning, 104(1):99–127, 2016.

19. Hernandez-Leal, P., Rosman, B., Taylor, M. E., Sucar, L. E., and Munoz de Cote, E.: A Bayesian approach for learning and tracking switching, non-stationary opponents. In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, pages 1315–1316. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

20. Ciregan, D., Meier, U., and Schmidhuber, J.: Multi-column deep neural networks for image classification. In 2012 IEEE Conference on Computer Vision and Pattern Recognition.

21. Farabet, C., Couprie, C., Najman, L., and LeCun, Y.: Learning hierarchical features for scene labeling. IEEE transactions on pattern analysis and machine intelligence, 35(8):1915–1929, 2013.

22. Krizhevsky, A., Sutskever, I., and Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

23. Boots, B., Siddiqi, S. M., and Gordon, G. J.: Closing the learning-planning loop with predictive state representations. The International Journal of Robotics Research, 30(7):954–966, 2011.

24. Littman, M. L. and Sutton, R. S.: Predictive representations of state. In Advances in neural information processing systems, pages 1555–1561, 2002.

25. Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J.: A robot that reinforcement-learns to identify and memorize important previous observations. In Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, volume 1, pages 430–435. IEEE, 2003.

26. Baxter, J. and Bartlett, P. L.: Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research, 15:319–350, 2001.

27. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.

28. Levine, S., Finn, C., Darrell, T., and Abbeel, P.: End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research, 17(1):1334–1373, 2016.

29. Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P.: Value iteration networks. In Advances in Neural Information Processing Systems, pages 2154–2162, 2016.

30. Hochreiter, S. and Schmidhuber, J.: Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

31. Hausknecht, M. and Stone, P.: Deep recurrent Q-learning for partially observable MDPs. In AAAI 2015 Fall Symposium, 2015.

32. Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al.: The predictron: End-to-end learning and planning. arXiv preprint arXiv:1612.08810, 2016.

33. Jonschkowski, R. and Brock, O.: End-to-end learnable histogram filters. In Workshop on Deep Learning for Action and Interaction at NIPS. Advances in neural information processing systems, 2016.

34. Haarnoja, T., Ajay, A., Levine, S., and Abbeel, P.: Backprop KF: Learning discriminative deterministic state estimators. In Advances in Neural Information Processing Systems, pages 4376–4384, 2016.

35. Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J.: Cognitive mapping and planning for visual navigation. arXiv preprint arXiv:1702.03920, 3, 2017.

36. Shankar, T., Dwivedy, S. K., and Guha, P.: Reinforcement learning via recurrent convolutional neural networks. In Pattern Recognition (ICPR), 2016 23rd International Conference on, pages 2592–2597. IEEE, 2016.

37. Harsanyi, J. C.: Games with incomplete information played by Bayesian players, i–iii part i. the basic model. Management science, 14(3):159–182, 1967.

38. Gmytrasiewicz, P. J. and Durfee, E. H.: Rational coordination in multi-agent environments. Autonomous Agents and Multi-Agent Systems, 3(4):319–350, 2000.

39. Fudenberg, D. and Levine, D. K.: The theory of learning in games, volume 2. MIT press, 1998.

40. Gilks, W. R., Richardson, S., and Spiegelhalter, D. J.: Introducing Markov chain Monte Carlo. Markov chain Monte Carlo in practice, 1:19, 1996.

41. Del Moral, P.: Non-linear filtering: interacting particle resolution. Markov processes and related fields, 2(4):555–581, 1996.

42. Liu, J. S. and Chen, R.: Sequential Monte Carlo methods for dynamic systems. Journal of the American statistical association, 93(443):1032–1044, 1998.

43. Gordon, N. J., Salmond, D. J., and Smith, A. F.: Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In IEE Proceedings F (Radar and Signal Processing), volume 140, pages 107–113. IET, 1993.

44. Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A.: A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866, 2017.

45. Bengio, Y., Courville, A., and Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013.

46. Deng, L., Yu, D., et al.: Deep learning: methods and applications. Foundations and Trends® in Signal Processing, 7(3–4):197–387, 2014.

47. Bengio, Y. et al.: Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1):1–127, 2009.

48. Giusti, A., Guzzi, J., Ciresan, D. C., He, F.-L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G., et al.: A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters, 1(2):661–667, 2016.

49. Kingma, D. P. and Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

50. Tieleman, T. and Hinton, G.: Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2):26–31, 2012.

51. Kurniawati, H., Hsu, D., and Lee, W. S.: SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In Robotics: Science and systems, volume 2008. Zurich, Switzerland., 2008.

52. Bubley, R. and Dyer, M.: Path coupling: A technique for proving rapid mixing in Markov chains. In Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on, pages 223–231. IEEE, 1997.

53. Han, Y. and Gmytrasiewicz, P. J.: Learning others' intentional models in multi-agent settings using interactive POMDPs. In The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence., pages 666–673, 2018.

54. Zhu, P., Li, X., Poupart, P., and Miao, G.: On improving deep reinforcement learning for POMDPs. arXiv preprint arXiv:1804.06309, 2018.

55. Russell, S. J. and Norvig, P.: Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,, 2016.

56. Aldous, D. J.: Exchangeability and related topics. In École d'Été de Probabilités de Saint-Flour XIII1983, pages 1–198. Springer, 1985.

57. Panella, A. and Gmytrasiewicz, P.: Interactive POMDPs with finite-state models of other agents. Autonomous Agents and Multi-Agent Systems, 31(4):861–904, 2017.

58. Abdi, H. and Williams, L. J.: Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4):433–459, 2010.

59. Bagnell, J. A., Kakade, S. M., Schneider, J. G., and Ng, A. Y.: Policy search by dynamic programming. In Advances in neural information processing systems, pages 831–838, 2004.

60. Doshi, P., Zeng, Y., and Chen, Q.: Graphical models for interactive POMDPs: representations and solutions. Autonomous Agents and Multi-Agent Systems, 18(3):376–416, 2009.

61. Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J.: Cognitive mapping and planning for visual navigation. ArXiv e-prints, February 2017.

62. Han, Y. and Gmytrasiewicz, P. J.: Learning others' intentional models in multi-agent settings using interactive POMDPs. In Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference 2017., pages 69–76, 2017.

63. Han, Y. and Gmytrasiewicz, P.: Decayed Markov chain Monte Carlo for interactive POMDPs. In Workshop on Deep Learning for Action and Interaction at NIPS, 2016.

64. Hernandez-Leal, P. and Kaisers, M.: Towards a fast detection of opponents in repeated stochastic games. In 1st Workshop on Transfer in Reinforcement Learning at AAMAS. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

65. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W.: ViZDoom: a Doom-based AI research platform for visual reinforcement learning. ArXiv e-prints, May 2016.

66. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

67. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, 2014.

68. Shankar, T., Dwivedy, S. K., and Guha, P.: Reinforcement learning via recurrent convolutional neural networks. ArXiv e-prints, January 2017.

69. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

70. Spaan, M. T. and Vlassis, N.: Perseus: randomized point-based value iteration for POMDPs. Journal of artificial intelligence research, 24:195–220, 2005.

71. Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al.: Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. arXiv preprint arXiv:1807.01281, 2018.

VITA

NAME	Yanlin Han
EDUCATION	B.A., Department of Network Engineering, Shandong University of Science and Technology, 2009 M.S., Department of Software Engineering, University of Science and Technology of China, 2011 M.S., Department of Computer Science, Clemson University, 2012 Ph.D., Department of Computer Science, University of Illinois at Chicago, 2018
EXPERIENCE	Research Assistant, Department of Computer Science, University of Illinois at Chicago, May 2013 - December 2018 Data Scientist Co-op, Motorola Solutions, January 2018 - December 2018 Teaching Assistant, Department of Computer Science, University of Illinois at Chicago, August 2013 - December 2017
PUBLICATIONS	Han, Y. and Gmytrasiewicz, P. J.: Learning others intentional models in multi-agent settings using interactive POMDPs. In Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, 2018 (to appear).

Han, Y. and Gmytrasiewicz, P. J.: Learning Others' Intentional Models in Multi-Agent Settings Using Interactive POMDPs. In the 28th Modern Artificial Intelligence and Cognitive Science Conference (MAICS), Indiana University/Purdue University - Fort Wayne, 2017.

Han, Y. and Gmytrasiewicz, P. J.: Decayed Markov chain Monte Carlo for interactive POMDPs. In NIPS Workshop on Deep Learning for Action and Interaction, Barcelona, Spain, 2016.

Han, Y. and Gmytrasiewicz, P. J.: Interactive Belief Update in Multi-agent Systems using Monte Carlo Sampling. In the Greater Chicago Area Systems Research Workshop (GCASR), University of Chicago, IL, 2016.