IPOMDP-Net: A Deep Neural Network for Partially Observable Multi-Agent Planning Using Interactive POMDPs

Yanlin Han, Piotr Gmytrasiewicz
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607

Abstract

This paper introduces the IPOMDP-net, a neural network architecture for multi-agent planning under partial observability. It embeds an interactive partially observable Markov decision process (I-POMDP) model and a QMDP planning algorithm that solves the model in a neural network architecture. The IPOMDP-net is fully differentiable and allows for end-to-end training. In the learning phase, we train an IPOMDP-net on various fixed and randomly generated environments in a reinforcement learning setting, assuming observable reinforcements and unknown (randomly initialized) model functions. In the planning phase, we test the trained network on new, unseen variants of the environments under the planning setting, using the trained model to plan without reinforcements. Empirical results show that our model-based IPOMDP-net outperforms the other state-of-the-art model-free network and generalizes better to larger, unseen environments. Our approach provides a general neural computing architecture for multi-agent planning using I-POMDPs. It suggests that, in a multi-agent setting, having a model of other agents benefits our decision-making, resulting in a policy of higher quality and better generalizability.

Introduction

Decision-making under partial observability is fundamentally important but computationally hard, especially in multi-agent settings. In partially observable multi-agent environments, an agent needs to deal with uncertainties from its own models as well as impacts from other agents actions caused by their models. To learn policies in such settings, one approach is to learn models and solve them through planning. If the models are known, it reduces to a planning problem and can be formulated as an interactive partially observable Markov decision process (I-POMDP) (Gmytrasiewicz and Doshi 2005). Although approximate algorithms have made progress on mitigating the computing complexity (Doshi and Gmytrasiewicz 2009; Han and Gmytrasiewicz 2018), solving I-POMDPs exactly is still computationally intractable for the worst case. Moreover, constructing I-POMDP models manually or learning them from observations remains difficult. An alternative approach is to learn policies directly, such as using model-free reinforcement learning methods, in which the rewards are usually assumed obtainable. While the model-free policy learning can be end-to-end, it lacks the model information for effective generalization (Karkus, Hsu, and Lee 2017). Particularly, in the multi-agent domain, other agents' actions impact the environment, which indirectly impact our policies. Therefore, we prefer a model-based approach, since having a model of other agents helps our reasoning about their actions and consequently benefits our decision making.

The unprecedented success of deep neural networks (DNNs) in supervised learning (Ciregan, Meier, and Schmidhuber ; Farabet et al. 2013; Krizhevsky, Sutskever, and Hinton 2012) provides new approaches to decision making under uncertainty. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been applied to tasks like Atari games (Mnih et al. 2015), robotics (Levine et al. 2016), and 2D path planning (Karkus, Hsu, and Lee 2017). 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 of convolutional layers for feature extraction and a fully connected layer for mapping features to actions, tackles many Atari games with the same network architecture (Mnih et al. 2015). DQN is inherently reactive and lacks explicit planning computation (Tamar et al. 2016). The deep recurrent Q-network (DRQN) extends DQN to the partially observable domain by replacing the fully connected layer with a recurrent long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) layer to integrate temporal information (Hausknecht and Stone 2015). Furthermore, the value iteration network (VIN) started to embed specific computation structures (Tamar et al. 2016), particularly the value iteration algorithm, in the network architecture and solves fully observable Markov decision processes (MDPs). The QMDP-net further embeds a POMDP model and a QMDP planning algorithm that solves the model in a RNN (Karkus, Hsu, and Lee 2017). 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.

In this work, we propose a neural network architecture, the IPOMDP-net, for multi-agent planning under par-

tial observability. We extends the QMDP-net to a multi-agent domain by combining an interactive POMDP (I-POMDP)(Gmytrasiewicz and Doshi 2005) model with a QMDP planning algorithm and embedding both in a recurrent neural network. We implement the IPOMDP-net on GPU-based computing devices and train it on problems of various sizes and dimensions. The training starts from randomly initialized weights, and is performed in a reinforcement learning fashion assuming the reward is obtainable. We then evaluate the trained IPOMDP-net by comparing its performance with another state-of-the-art model-free neural network. During the testing, we remove the obtainable reward assumption of all trained neural networks and test them on new unseen settings of the same problems. Therefore, they are using learned policies and acting based on observation and action sequences in the same way as the symbolic I-POMDP does. We show empirical results that our model-based IPOMDP-net outperforms the state-of-the-art model-free network in different tasks.

Our approach provides a general neural computing architecture for multi-agent planning using I-POMDPs. It combines the benefits of model-free learning and model-based planning. Compared with model-free networks, the IPOMDP-net trained on small problem sizes generalizes more effectively to larger difficult settings. It suggests that, in a multi-agent setting, having a model of other agents benefits our decision-making, resulting in a policy of higher quality and better generalizability.

Background

In this section, we will briefly introduce the underlying multi-agent planning model, the interactive POMDP (Gmytrasiewicz and Doshi 2005), that is encoded in our neural network architecture.

I-POMDP Framework

I-POMDPs generalize POMDPs (Kaelbling, Littman, and Cassandra 1998) to multi-agent settings by including models of other agents in the belief state space (Gmytrasiewicz and Doshi 2005). The resulting hierarchical belief structure represents an agent's belief about the physical state, belief about the other agents and their beliefs about others' beliefs, and can be nested infinitely in this recursive manner. For simplicity, we consider two interacting agents i and j. This formalism generalizes to more number of agents in a straightforward manner.

A computable finitely nested interactive POMDP of agent i , I-POMDP$_{i,l}$, is defined as:

$$I\text{-}POMDP_{i,l} = \langle IS_{i,l}, A, \Omega_i, T_i, O_i, R_i \rangle \tag{1}$$

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, $M_{j,l-1}$ is the set of possible models of agent j, and l is the strategy (nesting) level.

In this paper, we focus on a specific subset of model classes, the *intentional* models, which ascribe beliefs, preferences, and rationality in action selection to other agents. The *intentional* models, usually denoted as $\Theta_{j,l-1}$, of agent j at level $l-1$ is 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 agent j's belief nested to the level $(l-1)$, $b_{j,l-1} \in \Delta(IS_{j,l-1})$, and OC_j is j's optimality criterion. It can be rewritten as $\theta_{j,l-1} = \langle b_{j,l-1}, \hat{\theta}_j \rangle$, where $\hat{\theta}_j$ includes all elements of the intentional model other than the belief.

The $IS_{i,l}$ can be defined in an inductive manner:

$$\begin{aligned} &IS_{i,0} = S, && \theta_{j,0} = \{\langle b_{j,0}, \hat{\theta_j} \rangle : b_{j,0} \in \Delta(S)\} \\ &IS_{i,1} = S \times \theta_{j,0}, && \theta_{j,1} = \{\langle b_{j,1}, \hat{\theta_j} \rangle : b_{j,1} \in \Delta(IS_{j,1})\} \\ &\ldots\ldots && \\ &IS_{i,l} = S \times \theta_{j,l-1}, && \theta_{j,l} = \{\langle b_{j,l}, \hat{\theta_j} \rangle : b_{j,l} \in \Delta(IS_{j,l})\} \end{aligned} \tag{2}$$

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 all agents. Ω_i is the set of agent i's possible observations. $T_i : S \times A \times S \to [0, 1]$ is the transition function. $O_i : S \times A \times \Omega_i \to [0, 1]$ is the observation function. $R_i : IS_i \times A \to \mathbb{R}$ is the reward function.

Interactive Belief Update

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

$$\begin{aligned} &b^t_{i,l}(is^t) = Pr(is^t | b^{t-1}_{i,l}, a^{t-1}_i, o^t_i) \\ &= \alpha \sum_{is^{t-1}} b_{i,l}(is^{t-1}) \sum_{a^{t-1}_j} Pr(a^{t-1}_j | \theta^{t-1}_{j,l-1}) T(s^{t-1}, a^{t-1}, s^t) \times \\ &O_i(s^t, a^{t-1}, o^t_i) \sum_{o^t_j} O_j(s^t, a^{t-1}, o^t_j) \tau(b^{t-1}_{j,l-1}, a^{t-1}_j, o^t_j, b^t_{,l-1}) \end{aligned} \tag{3}$$

Conveniently, the equation above can be summarized as $b^t_{i,l} = SE(b^{t-1}_{i,l}, a^{t-1}_i, o^t_i)$.

Compared with POMDP, there are two major differences. First, the probability of other's actions given his models needs to be computed since the state now depends on both agents' actions (the second summation). Second, the modeling agent needs to update his beliefs based on the anticipation of what observations the other agent might get and how it updates (the third summation).

While exact interactive belief update is intractable, there are sampling-based approximations using customized interactive version of particle filters (Doshi and Gmytrasiewicz 2009; **?**), which will be embedded in the IPOMDP-net in a neural analogy.

The value iteration for I-POMDP is performed on interactive belief states in the following way:

$$\begin{aligned} V(\theta_{i,l}) &= \max_{a_i \in A_i} Q(\theta_{i,l}, a_i) \\ &= \max_{a_i \in A_i} \Big\{ \sum_{is \in IS} b_{i,l}(is) ER_i(is, a_i) + \\ &\gamma \sum_{o_i \in \Omega_i} P(o_i | a_i, b_{i,l}) V(\langle SE(b_{i,l}, a_i, o_i), \hat{\theta}_i \rangle) \Big\} \end{aligned} \tag{4}$$

where $ER_i(is, a_i) = \sum_{a_j} R_i(is, a_i, a_j) Pr(a_j | \theta_{j,l-1})$.

Then the optimal action, a_i^*, for an infinite horizon criterion with discounting, is part of the set of optimal actions, $OPT(\theta_i)$, for the belief state:

$$OPT(\theta_{i,l}) = \arg\max_{a_i \in A_i} Q(\theta_{i,l}, a_i) \quad (5)$$

IPOMDP-Net

Overview

The IPOMDP-net is a neural analogy of the I-POMDP framework. 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 (Karkus, Hsu, and Lee 2017), it combines a parameterized model with an approximate algorithm that solves the model in a single, differentiable neural network. But 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 section, 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 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. In the case that both agents' models are known, model parameters are preassigned as weights and this task reduces to a multi-agent planning problem.

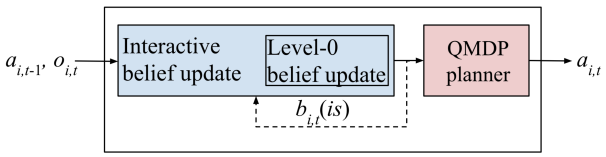


Figure 1: IPOMDP-net architecture overview. It embeds an interactive belief update algorithm and the QMDP planner, the hidden state encodes the interactive belief of agent i.

An IPOMDP-net consists of three main network modules as shown in Figure 1. The first two modules (blue boxes) perform interactive belief update using a customized particle filter for intentional models (Han and Gmytrasiewicz 2018). 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) represents the QMDP algorithm, which chooses the action given the current belief. Besides being a planning network, all modules of IPOMDP-net are differentiable, allowing the entire network to be trained end-to-end.

Network Architecture

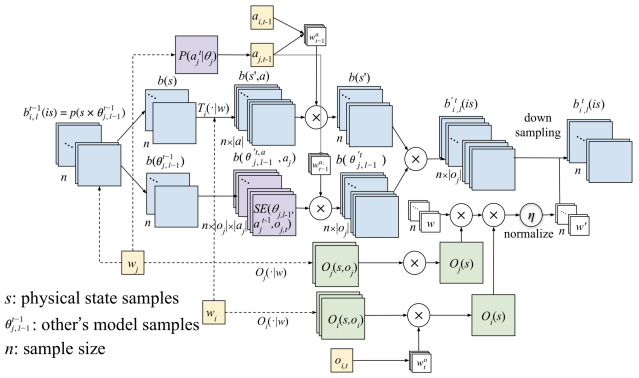
The intuition behind embedding the I-POMDP model and the QMDP algorithm in a single, differentiable neural network is the neural analogy of linear and maximum operations used in the related computations. Namely, the matrix multiplications and summations can be represented by convolutional layers and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

Below we will give some details on the individual modules. For simplicity, assume there are two agent i and j in the game and the strategy level is 1. Consider the two-agent tiger game (Gmytrasiewicz and Doshi 2005), which generalizes the classic single agent tiger game (Kaelbling, Littman, and Cassandra 1998) to multi-agent settings. 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 rewards for getting the gold or penalties 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. In the I-POMDP formation, it is defined as: $IS_{i,1} = S \times \Theta_{j,0}$, where $S = \{$tiger on the left (TL), tiger on the right (TR)$\}$ and $\Theta_{j,0} = \{< b_j(s), A_j, \Omega_j, T_j, O_j, R_j, OC_j >\}$; $A = A_i \times A_j$ is the set of joint actions, $\{$listen (L), open left door (OL) and open right door(OR)$\} \times \{$L, OL, OR$\}$; Ω_i: $\{$growl from left (GL) or right (GR)$\} \times \{$creak from left (CL), right (CR) or silence (S)$\}$; $T_i = T_j : S \times A_i \times A_j \times S \to [0, 1]$; $O_i : S \times A_i \times A_j \times \Omega_i \to [0, 1]$; $R_i : IS \times A_i \times A_j \to \mathbb{R}$.

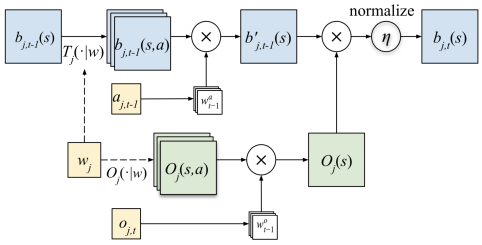
The IPOMDP-net works on a sampling-based representation of interactive belief state $IS_{i,1} = S \times \Theta_{j,0}$. For the tiger game, a sample of physical state can be simply denoted using one-hot vectors, for example [1,0] represents s=TL. A sample of j's model can be represented as a vector of 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 belief (0.5), and transition $(0.67, 0.5)$, observation $(0.85, 0.5)$, and reward functions $(-1, -100, 10)$ (see (Han and Gmytrasiewicz 2018) for details). 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 (Han and Gmytrasiewicz 2018). It consists 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.

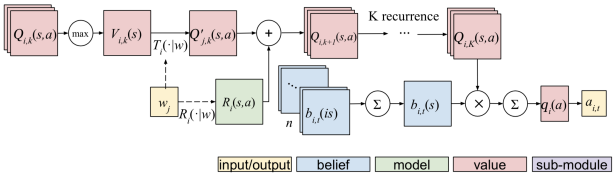
The interactive belief update module maps agent i's interactive belief, action, and observation to a next belief, $b_{i,l}^t = SE(b_{i,l}^{t-1}, a_i^{t-1}, o_i^t)$ (Equation 3). It can be decomposed into two major steps: when agent i performs an action a_i^{t-1}, and given j performs a_j^{t-1}, it predicts the belief state (Equation 6); then when i perceives an observation o_i^t, it cor-



(a) Interactive belief update module



(b) Level-0 belief update module



(c) QMDP planner module

Figure 2: IPOMDP-net consists of three modules

rects and normalizes the prediction (Equation 7).

$$\hat{b}^t_{i,l}(is^t) = \sum_{is^{t-1}} b_{i,l}(is^{t-1}) \sum_{a_j^{t-1}} Pr(a_j^{t-1}|\theta^{t-1}_{j,l-1}) \times T(s^{t-1}, a^{t-1}, s^t) \sum_{o_j^t} O_j(s^t, a^{t-1}, o_j^t) \times \tau(b^{t-1}_{j,l-1}, a_j^{t-1}, o_j^t, b^t_{,l-1}) \quad (6)$$

$$b^t_{i,l}(is^t) = \alpha \sum_{a_j^{t-1}} O_i(s^t, a^{t-1}, o_i^t) \hat{b}^t_{i,l}(is^t) \quad (7)$$

Equation 6 is implemented using convolutional layers and sub-modules. Firstly, 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 of the convolutional layer. 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 $is^{t-1}_{i,1}$, the samples of the other agent j's model $\theta^{t-1}_{j,0}$, are the inputs into the level-0 belief update module ($b^t_{j,0} = SE(b^{t-1}_{j,0}, a^{t-1}_j, o^t_j)$ in Figure 2(b)). They 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 $\tau(b^{t-1}_{j,l-1}, a^{t-1}_j, o^t_j, b^t_{,l-1})$ function in Equation 6. 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-models being used.

Since $\hat{b}^t_{i,1}(s^t, a)$ after convolution encodes predicted physical belief after taking each of the joint actions, $a \in A = A_i \times A_j$, we need to select the belief corresponding to the last joint action. In Figure 1(a), notice that the w_i and w_j contain all the parameters of both i and j's models, thus we compute j's optimal actions according to j's model, $P(a^t_j|\theta^{t-1}_j)$. $P(a^t_j|\theta^{t-1}_j)$ can be any single-agent POMDP solver, in this case we plug in a pretrained QMDP-net (Karkus, Hsu, and Lee 2017) to make the entire network end-to-end. Then we use the soft indexing similar to the one used in QMDP-net (Karkus, Hsu, and Lee 2017), where w^a_{t-1} is the indexing vector, a distribution over A. Then we weight $\hat{b}^t_{i,1}(s^t, a)$ by w^a_{t-1}:

$$\hat{b}^t_{i,l}(s^t) = \sum_{a \in A} \hat{b}^t_{i,l}(s^t, a) w^a_{t-1} \quad (8)$$

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

$$\hat{b}^t_{i,l}(\theta^t_{j,l-1}) = \sum_{a_j \in A_j} \hat{b}^t_{i,l}(\theta^t_{j,l-1}, a_j) w^{a_j}_{t-1} \quad (9)$$

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 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 observations, it is a $D_1 \times D_2 \times |O_i|$ tensor. The initial weights of belief samples $w(b_i^0)$ is uniformly initialized as $1/|n|$ and then weighted by anticipated observations of j and actual observation of i. We select the observation function corresponding to i's last observation using soft indexing again:

$$O_i(s) = \sum_{a \in A} O_i(s, o_i) w_{t-1}^o \tag{10}$$

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 11 and 12, the two classic steps are the prediction through transition and the correction using observation.

$$\hat{b}_j^t(s^t) = \sum_{s^{t-1} \in S} T(s^{t-1}, a_j^{t-1}, s) b_j^{t-1}(s^{t-1}) \tag{11}$$

$$b_j^t(s^t) = \alpha O(s^t, a_j^{t-1}, o_j^t) \hat{b}_j^t(s^t) \tag{12}$$

This module is implemented almost identically to the belief update (filter) module in the QMDP-net (Karkus, Hsu, and Lee 2017), 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 refrain from repeating the explanation, but the basic idea is to represent the matrix multiplication in Equation 11 as a convolutional layer, use soft indexing to select a_j and o_j, and make element-wise multiplication in Equation 12.

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.

$$Q_{i,k+1}(s, a_i) = R_i(s, a_i) + \gamma \sum_{s'} T_i(s, a, s') V_{i,k}(s') \tag{13}$$

$$V_{i,k}(s) = \max_{a_i} Q_{i,k}(s, a_i) \tag{14}$$

The value iteration in Equation 13 and 14 is implemented using convolutional and max pooling layers(Tamar et al. 2016; **?**). The $Q_i(s, a_i)$ is a $D_1 \times D_2 \times |A_i|$ tensor. Equation 13 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 14 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 13 and 14 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 16). But before that, since i's interactive belief contains j's models as well, we need to marginalize over models of j (Equation 15). Finally, we select the action that has the highest q-values.

$$b_{i,t}(s) = \sum_{\theta_j} b_{i,t}(is) \tag{15}$$

$$Q_i(b_i, a_i) = \sum_s Q_{i,K}(s, a_i) b_{i,t}(s) \tag{16}$$

Training Algorithm

We train the IPOMDP-net in a reinforcement learning setting following the similar way in DQN (Mnih et al. 2015) (Algorithm 1). Due to partial observability, we 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 obtainable and agent i's belief is computed in the belief update module. We have also used the ϵ-greedy strategy to ensure adequate explorations of the belief space, especially when the underlying planner is QMDP.

Algorithm 1: IPOMDP-net Training

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 $o_{i,t+1}$, 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 $y_m =$
$\begin{cases} r_{i,m} & \text{if i+1 is terminal step} \\ r_{i,m} + \beta \max_{a_i} \hat{Q}(b_{i,m+1}, a_i'; w^-) & \text{otherwise} \end{cases}$
    perform gradient descent on: $(y_m - Q(b_{i,m}, a_{i,m}; w))^2$

Model-Free Networks for Comparison

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

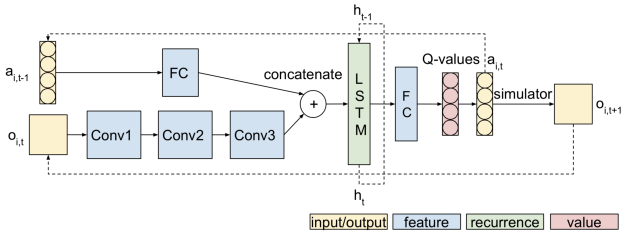


Figure 3: One time slice of ADRQN. Inputs are $a_{i,t-1}$ and $o_{i,t}$, outputs are $a_{i,t}$ and $o_{i,t+1}$. FC stands for fully connected layer. Conv stands for convolutional layers. Output of LSTM layer h_t will be input into LSTM in the next time slice. Actual hyper parameters vary on different problems.

deep recurrent Q-network (ADRQN) for single-agent domain (Zhu et al. 2018).

As shown in Figure 3, 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. These latent states integrate the information contained in action and observation histories. It has been shown that the ADRQN outperforms the ARQN (Zhu et al. 2018), which outperforms the DQN (Mnih et al. 2015). Thus, it is one of the state-of-the-art model-free networks for sequential decision making problems.

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 ADRQN is that weights in the IPOMDP-net encode both agents' model parameters, but in ADRQN the weights are from another parameter space.

Experiments

Experimental setup

The goal of the experiments is to verify 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. Since IPOMDP-net is a neural approximation to the symbolic I-POMDP, we also want to know how close this approximation is in terms of planning performance.

We firstly test the planning performance of IPOMDP-net by comparing it with its symbolic counterpart. We initialize weights of the IPOMDP-net using true parameters of model functions and test it in various but fixed environments. In this planning setting, the reinforcements are unobservable and model functions are known. We show related results in Table 1.

Then we apply the same IPOMDP-net architecture in model-based reinforcement learning (RL) problems, where there are two phases of experiments: training and testing. The training phase is for different neural networks (IPOMDP-net and ADRQN) to be trained, starting from randomly initialized weights and observable reinforcements, so that the converged networks are approximations to true policy functions. The testing phase is to evaluate the trained networks in a planning setting where the rewards are unobservable. We show related results in Table 2.

Therefore, we designed experiments of five problems in two categories. In the first category, we use the two-agent tiger game (Gmytrasiewicz and Doshi 2005) and UAV problem (Doshi and Gmytrasiewicz 2009), 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 (Russell and Norvig 2016) 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 model-based IPOMDP-net and the IPOMDP-net 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.

Results and Discussions

Table 1: Average results for planning tasks. The IPOMDP-net with preassigned weight performs almost the same as it symbolic I-POMDP counterpart. The results are averaged over 50 random runs.

	IPOMDP-net	I-POMDP	I-POMDP SARSOP
Tiger	2.25 ± 0.11	2.26 ± 0.09	2.32 ± 0.15
UAV	9.10 ± 0.39	9.09 ± 0.45	9.27 ± 0.52
Maze 4×4	0.17 ± 0.05	0.16 ± 0.06	0.19 ± 0.04
Maze 10×10	-0.53 ± 0.09	-0.56 ± 0.08	-0.49 ± 0.07
Maze 16×16	-0.99 ± 0.10	-0.96 ± 0.09	-0.81 ± 0.10

In Table 1, we use known model parameters to preassign IPOMDP-net weights, the corresponding symbolic I-POMDP (with QMDP planner) is shown in the second column. In the last column, we also report additional results on the symbolic I-POMDP using SARSOP planner (Kurniawati, Hsu, and Lee 2008) instead of QMDP, which represents the best performance that a symbolic approach can achieve now. We will look into ways of implementing SARSOP or other planners in IPOMDP-net, which generally requires more sophisticated network design.

We see that in all problems, the performance of preinitialized IPOMDP-net is almost identical to the symbolic I-POMDP, which is to be expected as they are the neural and symbolic implementations of the same I-POMDP framework.

In Table 2, we report the average rewards in Tiger, UAV, and three Maze variations. Wee see that the performance

Table 2: Average rewards for different RL problems. The networks start from random weights. The results are averaged over 50 random runs.

(a) The learning and testing environments are fixed for Tiger and UAV. The performance difference is small.

	Fixed environments	
	Tiger	UAV
ADRQN	1.29 ± 0.35	8.98 ± 0.97
IPOMDP-net w/ trained weights	1.32 ± 0.29	9.19 ± 0.72

(b) 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.

	Random environments - Maze		
	4×4	10×10	16×16
ADRQN	0.12 ± 0.08	-0.73 ± 0.17	-1.58 ± 0.39
IPOMDP-net trained	0.18 ± 0.05	-0.52 ± 0.10	-0.88 ± 0.21

of the model-free ADRQN degrades faster as maps size increases from 4×4 to 16×16.

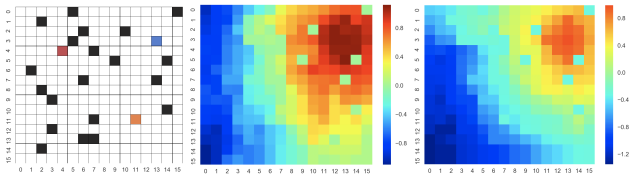
IPOMDP-net learns policies that generalize to new environments. In the fixed Tiger and UAV environments (Table 2(a)), the model-free ADRQNs have comparable performance to the IPOMDP-net. The reason is that in a fixed environment, a network may directly learn the mapping from features to policy. In contrast, the IPOMDP-net learns a model for planning, i.e. generating a near-optimal policy for arbitrary 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 2(b), the average rewards of IPOMDP-net are higher than the ADRQN.

IPOMDP-net policies learned in small environments transfers directly to larger ones. 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 and 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. We see that the IPOMDP-net is significantly better than the model-free ADRQN, the performance of ADRQN degrades fast when the maze size increases.

IPOMDP-net learns an overall better policy instead of a "true" model. It makes sense that the learned model should be the ground truth if the embedded I-POPMDP algorithm is exact. Since our 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 reason is that the end-to-end training gives IPOMDP-net the opportunity to learn an "incorrect" but useful model that compensates the limitation of the approximation algorithm, the QMDP.

Visualization

We visualize the learned value function of agent i for the 16×16 Maze problem. In Figure 4(b), we see 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 4(c), since j is modeled as a level-0 POMDP agent, he has no clue about i's existence. Thus, in j's reward function, states close to the goal have high values.



(a) (b) (c)

Figure 4: 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. Black squares are obstacles, red is i's location, orange is j's location, and blue is the target location.

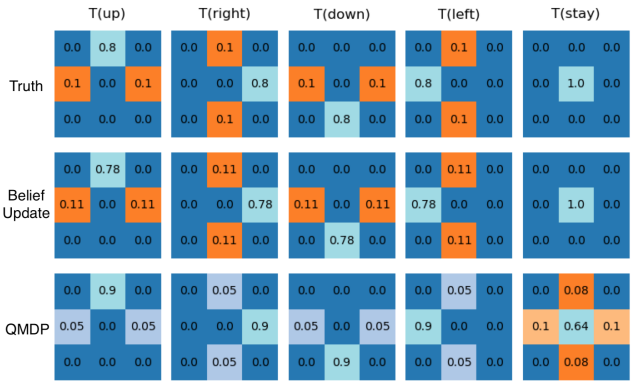


Figure 5: The learned transitions in belief update module and QMDP module are different. The first row is the ground truth. The second row is the transition in belief update module. The third row is the transition in QMDP module.

As an example of learned "incorrect" models, we also visualize the learned transition functions in interactive belief update and QMDP planning modules. We see that in Figure 5, 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.

Conclusion

We have described the IPOMDP-net, a neural network architecture for multi-agent planning under partial observability. It combines model-free learning and model-based planning by embedding an I-POMDP model and a QMDP planning algorithm in a network learning architecture. We show its effectiveness, performance, and generalizability on various problems. Our approach provides a general, model-based neural computing architecture for multi-agent planning.

One future research direction is to implement the exact I-POMDP value iteration 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. Currently one major constraint of our work is the scalability when there are multiple agents or the input size / problem state is huge. This may lead to another direction, which is an attention mechanism (Tamar et al. 2016) to reduce the effective number of network parameters, because in many problems, the importance and effects of neighboring states might be contextually dependent on what is the current state.

References

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*.

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

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

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

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

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

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

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

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

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

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

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

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. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

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

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

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