QMDP-Net: Deep Learning for Planning under Partial Observability

Peter Karkus[1,2] **David Hsu**[1,2] **Wee Sun Lee**[2]

[1]NUS Graduate School for Integrative Sciences and Engineering
[2]School of Computing

National University of Singapore
`{karkus, dyhsu, leews}@comp.nus.edu.sg`

Abstract

This paper introduces the QMDP-net*, a neural network architecture for planning under partial observability. The QMDP-net combines the strengths of model-free learning and model-based planning. It is a recurrent policy network, but it represents a policy by connecting a model with a planning algorithm that solves the model, thus embedding the solution structure of planning in a network learning architecture. The QMDP-net is fully differentiable and allows end-to-end training. We train a QMDP-net in a set of different environments so that it can generalize over new ones and "transfer" to larger environments as well. In preliminary experiments, QMDP-net showed strong performance on several robotic tasks in simulation. Interestingly, while QMDP-net encodes the QMDP algorithm, it sometimes outperforms the QMDP algorithm in the experiments, because of QMDP-net's increased robustness through end-to-end learning.*

1 Introduction

Decision making under uncertainty is of fundamental importance in many applications, but is computationally hard, especially under partial observability [21]. In a partially observable world, the agent cannot determine its state exactly based on the current observation; to plan optimal actions, it must integrate information over the past history of actions and observations. See Fig. 1 for an example. In the model-based approach, we may formulate the problem as a *partially observable Markov decision process* (POMDP). Solving POMDPs exactly is computationally intractable in the worst case [21]. Approximate POMDP algorithms have made dramatic progress on solving large-scale POMDPs [16, 22, 26, 29, 32]; however, constructing POMDP models manually or learning POMDP models from data remain difficult. In the model-free approach, we directly search for an optimal solution within a policy class. If we do not restrict the policy class, the difficulty is data and computational efficiency. We may choose a parameterized policy class. The effectiveness of policy search is then constrained by this a priori choice.

Deep neural networks have brought unprecedented success in many domains [15, 20, 27] and provide a distinct new approach to decision making under uncertainty. The deep Q-network (DQN), a convolutional neural network (CNN) together with a fully connected layer, has successfully tackled many Atari games with complex visual input [20]. Replacing the post-convolutional fully connected layer of DQN by a recurrent LSTM layer enables it to deal with partial observaiblity [9]. This approach, however, ignores the underlying sequential-decision nature of planning.

We introduce *QMDP-net*, a neural network architecture for planning under partially observability. It combines the strengths of model-free and model-based approaches. A QMDP-net is a recurrent policy network, but represents a policy by connecting a POMDP model with an algorithm that solves the model, thus embedding the solution structure of planning in the network architecture. Specifically, our network uses QMDP [17], a simple, but fast approximate POMDP algorithm, though other more sophisticated POMDP algorithms would also be possible.

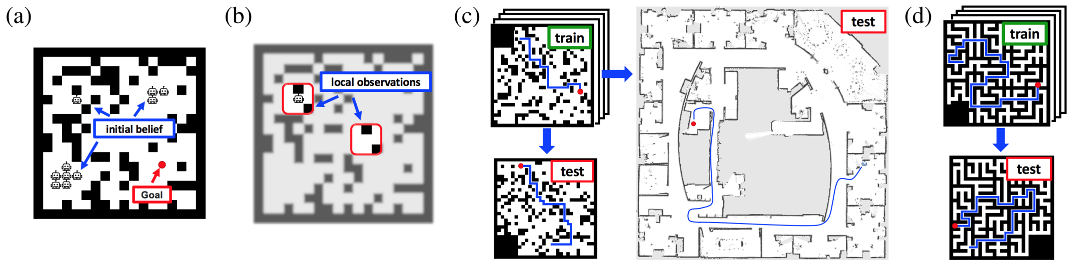

Fig. 1: A robot learning to navigate in partially observable grid worlds. (a) The robot has a map. It has a belief over the initial state, but does not know the exact initial state. (b) Local observations are ambiguous and are insufficient to determine the exact state. (c, d) A policy trained on expert demonstrations in a set of randomly generated environments generalizes to a new environment. It also "transfers" to a much larger real-life environment, represented as a LIDAR map [11].

A QMDP-net consists of two main network modules (Fig. 2). One is a Bayesian filter that integrates the history of an agent's actions and observations into a *belief*, i.e. a probabilistic estimate of the agent's state. The other is the QMDP algorithm, which chooses the action given the current belief. Both components are differentiable, allowing the entire network to be trained end-to-end.

We train a QMDP-net on expert demonstrations in a set of randomly generated environments. The trained policy generalizes to new environments and also "transfers" to more complex environments (Fig. 1c–d). Preliminary experiments show that QMDP-net outperformed state-of-the-art network architectures on several robotic tasks in simulation. It successfully solved difficult POMDPs that require reasoning over many time steps, such as the well-known Hallyway2 domain [17]. Interestingly, while QMDP-net encodes the QMDP algorithm, it sometimes outperformed the QMDP algorithm in our experiments, because of QMDP-net's increased robustness through end-to-end learning.

2 Background

2.1 Planning under uncertainty

A POMDP is formally defined as a tuple (S, A, O, T, Z, R), where S, A and O are the state, action, and observation space, respectively. The state-transition function $T(s, a, s') = P(s'|s, a)$ defines the probability of the agent being in state s' after taking action a in state s. The observation function $Z(s, a, o) = p(o|s, a)$ defines the probability of receiving observation o after taking action a in state s. The reward function $R(s, a)$ defines the immediate reward for taking action a in state s.

In a partially observable world, the agent does not know its exact state. It maintains a *belief*, which is a probability distribution over S. The agent starts with an initial belief b_0 and updates the belief at each time step with a Bayesian filter: $b_t(s') = \tau(b_{t-1}, a_t, o_t) = \eta O(s', a_t, o_t) \sum_{s \in S} T(s, a_t, s') b_{t-1}(s)$, where η is a normalizing constant. The belief b_t at time t integrates information from the *entire* past history $(a_1, o_1, a_2, o_2, \ldots, a_t, o_t)$ for decision making. POMDP planning seeks a *policy* π that maximizes its *value*, i.e., the expected total discounted reward: $V_\pi(b_0) = \mathbb{E}\big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_{t+1}) \bigm| b_0, \pi\big)$, where s_t is the state at time t, $a_{t+1} = \pi(b_t)$ is the action that the policy π chooses at time t, and $\gamma \in (0, 1)$ is a discount factor.

2.2 Related work

To learn policies for decision making in partially observable domains, one approach is to learn models [6, 18, 23] and solve the models through planning. An alternative is to learn policies directly [2, 5]. Model learning is usually not end-to-end. While policy learning can be end-to-end, it does not exploit model information for effective generalization. Our proposed approach combines model-based and model-free learning by embedding a model and a planning algorithm in an recurrent neural network (RNN) that represents a policy and training the network end-to-end.

RNNs have been used earlier for learning in partially observable domains [4, 9, 10]. In particular, Hausknecht and Stone extended DQN [20], a convolutional neural network (CNN), by replacing its post-convolutional fully connected layer with a recurrent LSTM layer [9]. Similarly, Mirowski et al. [19] considered learning to navigate in partially observable 3-D mazes. The learned policy generalizes over different goals, but in a fixed environment. Instead of using the generic LSTM, our approach embeds structure specific to sequential decision making in the network architecture and aims to learn a policy that generalizes to new environments.

The idea of embedding specific computational structures in the neural network architecture has been gaining attention recently. Tamar et al. implemented value iteration in a neural network, called Value Iteration Network (VIN), to solve Markov decision processes (MDPs) in fully observable domains, where an agent knows its exact state and does not require filtering [31]. It does not address the issue of partial observability, which drastically increases the computational complexity of decision making [21]. Haarnoja et al. [8] and Jonschkowski and Brock [14] developed end-to-end trainable Bayesian filters for probabilistic state estimation. Silver et al. introduced Predictron for value estimation in Markov reward processes [28]. They do not deal with decision making or planning. Both Shankar et al. [25] and Gupta et al. [7] addressed planning under partial observability. The former focuses on learning a model rather than a policy. The learned model is trained on a fixed environment and does not generalize to new ones. The latter proposes a network learning approach to robot navigation in an unknown environment, with a focus on mapping. Its network architecture contains a hierarchical extension of VIN for planning and thus does not deal with partial observability during planning. We propose to impose the POMDP model and computation structure priors on the entire network architecture in order to handle partial observability effectively.

3 Overview

We want to learn a policy that enables an agent to act effectively in a diverse set of partially observable stochastic environments. Consider, for example, the robot navigation domain in Fig. 1. The robot agent does not observe its own location directly, but estimates it based on noisy readings from a laser range finder. The environments may correspond to different buildings. The agent has access to building maps, but does not have models of its own dynamics and sensors. While the buildings may differ significantly in their layouts, the underlying reasoning required for effective navigation is similar in all buildings. After training the robot in a few buildings, we want to place the robot in a new building and have it navigate effectively to a specified goal.

Formally, the agent learns a policy for a parameterized class of tasks in partially observable stochastic environments: $\mathcal{W}_\Theta = \{W(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}$, where Θ is the set of all parameter values. The parameter value $\boldsymbol{\theta}$ captures a wide variety of task characteristics that vary within the class, including environments, goals, and agents. In our robot navigation example, $\boldsymbol{\theta}$ encodes a map of the environment, a goal, and a belief over the robot's initial state. We assume that all tasks in $\mathcal{W}(\boldsymbol{\theta})$ share the same state space, action space, and observation space, though the assumption can be relaxed to allow dependence of the state space on $\boldsymbol{\theta}$ (see Section 5). The agent does not have prior models of its own dynamics, sensors, or task objectives. After training on tasks for some subset of values in Θ, the agent learns a policy that solves $W(\boldsymbol{\theta})$ for any given $\boldsymbol{\theta} \in \Theta$.

A key issue here is a general representation of a policy for $\mathcal{W}_\Theta$, without knowing the specifics of $\mathcal{W}_\Theta$ or its parametrization. We introduce QMDP-net, a recurrent policy network. QMDP-net represents a policy by connecting a parameterized POMDP model with an approximate POMDP algorithm and embedding both in a single, differentiable neural network. Embedding the model allows the policy to generalize over $\mathcal{W}_\Theta$ effectively. Embedding the algorithm allows us to train the entire network end-to-end and learn an internal model that compensates for the limitations of the approximate algorithm.

Let $M(\boldsymbol{\theta}) = (S, A, O, f_T(\cdot|\boldsymbol{\theta}), f_Z(\cdot|\boldsymbol{\theta}), f_R(\cdot|\boldsymbol{\theta}))$ be the embedded POMDP model, where S, A and O are the state space, action space, observation space designed manually for $\mathcal{W}_\Theta$ and $f_T(\cdot|\cdot), f_Z(\cdot|\cdot), f_R(\cdot|\cdot)$ are the state-transition, observation, and reward function to be learned from data. It may appear that a perfect answer to our learning problem would have $f_T(\cdot|\boldsymbol{\theta}), f_Z(\cdot|\boldsymbol{\theta}),$ and $f_R(\cdot|\boldsymbol{\theta})$ represent the "true" underlying models of dynamics, observation, and reward for the task $W(\boldsymbol{\theta})$. This is true only if the embedded POMDP algorithm is exact, but not true in general. The agent may learn an alternative model to mitigate an approximate algorithm's limitations and obtain an overall better policy. In this sense, while QMDP-net embeds a POMDP model in the network architecture, it aims to learn a good policy rather than a "correct" model.

The embedded algorithm in a QMDP-net consists of two modules (Fig. 2). One encodes a Bayesian filter, which performs state estimation by integrating the past history of agent actions and observations

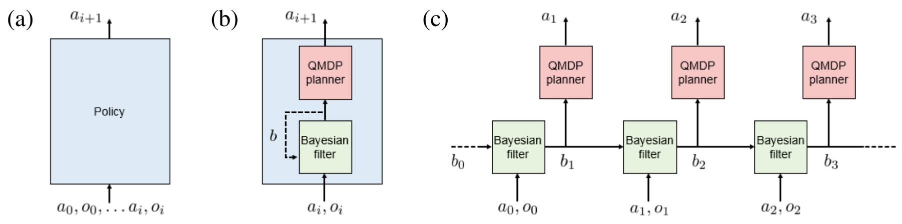

Fig. 2: QMDP-net architecture. (a) A policy maps a history of actions and observations to a new action. (b) QMDP-net is a recurrent policy network that imposes structure priors for sequential decision under partial observability: it embeds a Bayesian filter and a POMDP planner. The hidden state of the RNN encodes the belief for POMDP planning. (c) QMDP-net unfolded in time.

into a belief. The other encodes QMDP, a simple, but fast approximate POMDP planner [17]. QMDP chooses the agent's action by solving the corresponding fully observable Markov decision process (MDP) and performing one-step look-ahead search on the MDP values weighted by the belief.

We evaluate the proposed network architecture in an imitation learning setting. We train on a set of expert trajectories with randomly chosen parameter values in Θ and test with new parameter values. An expert trajectory consist of a sequence of demonstrated actions and observations $(a_1, o_1, a_2, o_2, \ldots)$ for some $\boldsymbol{\theta} \in \Theta$. The agent does not have access to the underlying states or ground-truth beliefs along the trajectory during the training. We define loss as the cross entropy between predicted and demonstrated action sequences. We implemented QMDP-net in Tensorflow and used RMSProp for training. See Appendix C.6 for details. We plan to release the source code in the future.

4 QMDP-net

QMDP-net is a recurrent policy network that encodes a parameterized POMDP model $M(\boldsymbol{\theta}) = (S, A, O, T = f_T(\cdot|\boldsymbol{\theta}), Z = f_Z(\cdot|\boldsymbol{\theta}), R = f_R(\cdot|\boldsymbol{\theta}))$ and selects actions by approximating a solution to this model. The input to the network is a task parameter $\boldsymbol{\theta}$ and a sequence of observations and past actions. The output is a sequence of desired actions. Observations are partial observations of the dynamic agent state, $\boldsymbol{\theta}$ is a full observation of the static environment. In our experiments observations are vectors (e.g. LIDAR readings) and $\boldsymbol{\theta}$ is an image (e.g. a map), though other representations would be possible. When the policy is executed, the action output at step i is fed to the input of step $i+1$ along with the observation received after executing the action.

We choose the structure of the embedded model M based on domain knowledge. We do not know the parameters of M. Instead, we expect that a useful M will automatically emerge through end-to-end training. For convenience we use $\mathbb{W}$ to refer to a POMDP that describes the underlying decision making task, though we do not need to know the parameters nor the structure of $\mathbb{W}$. The space of actions and observations can differ in $\mathbb{W}$ and M. We handle this by learning appropriate mappings (f_A, f_O, f_π). There are two components in the QMDP-net architecture: a filter and a planner.

Filter module. The filter module (Fig. 3a) implements a Bayesian filter that propagates the belief. It maps from a belief, action, and observation to a next belief, $b_{i+1} = f(b_i|a_i, o_i)$. The first step in a Bayesian filter accounts for transitions, $b'_i(s) = \sum_{s'} T(s, a_i, s')b_i(s')$, while the second step incorporates observations, $b_{i+1}(s) = Z(s, a_i, o_i)b'_i(s)/\eta$, where o_i is the observation received after taking action a_i and η is a normalization factor. We implement the filter using a neural network representation of M. For the ease of discussion let us assume an $N \times N$ state space for now. We represent a belief, $b_i(s)$, by an $N \times N$ image,[1] where pixels correspond to the probability of being in the corresponding state, s.

The first step of the Bayesian filter is a convolution, f_T, which corresponds to the transition function in the embedded model. The output of f_T is an $N \times N \times |A|$ image, $b'_i(s, a)$, where each channel of the image corresponds to the belief after taking action $a \in A$. We select the appropriate channel of $b'_i(s, a)$ based on the action input of the filter a_i. While $b'_i(s, a)$ is defined in the internal model M, action inputs are defined in $\mathbb{W}$. We learn a mapping, f_A, from action inputs to a vector w^{a_i}, an indexing distribution over actions in M. We select the appropriate channel of $b'_i(s, a)$ by "soft indexing", i.e. we compute the sum of $b'_i(s, a)$ along its channels weighted by w^{a_i}, $b'^{a}_i(s) = \sum_a b'_i(s, a)w^a_i$.

[1]We can refer to matrices as n-dimensional images without loosing generality.

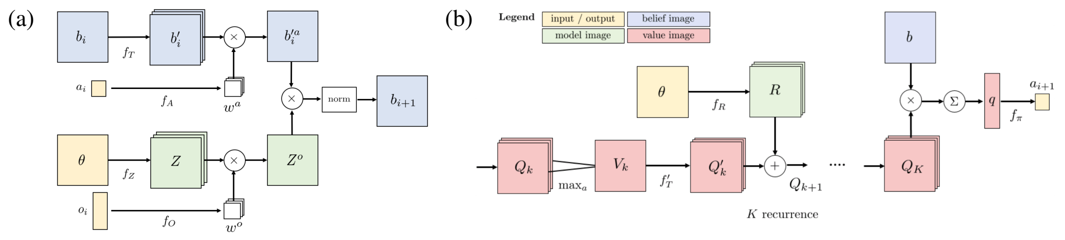

Fig. 3: A QMDP-net is composed of two modules. (a) The Bayesian filter module incorporates an action-observation history into a belief. (b) The QMDP planner module solves an MDP by value iteration and chooses an action that maximizes the MDP state-action values weighted by the belief.

The second step incorporates observations through a learned observation model $Z = f_Z(s, o|\boldsymbol{\theta})$, an $N \times N \times |O|$ image that represents the probability of receiving o in state s. The observation model Z depends on the task instance $\boldsymbol{\theta}$, e.g. the location of obstacles for a navigation task. We obtain Z through a mapping from $\boldsymbol{\theta}$, $Z = f_Z(s, o|\boldsymbol{\theta})$. In this paper we realize f_Z by a CNN. We need to select the appropriate channel of Z given the observation input o_i. Again, observation inputs are defined in $\mathbb{W}$ and not in M. We map from o_i to an indexing vector, $w_i^o = f_O(o_i)$; and apply soft indexing, $Z^o(s) = \sum_o Z(s, o) w_i^o$.

Finally, we multiply the belief image $b_i'^a(s)$ and $Z^o(s)$ element-wise and normalize over states to obtain the updated belief, $b_{i+1}(s)$. In our setting the initial belief for each task instance is encoded in $\boldsymbol{\theta}$. We can initialize the belief in QMDP-net through an additional mapping, $b_0 = f_B(\boldsymbol{\theta})$.

Planner module. The planner implements value iteration and maps from beliefs to actions by weighting Q values with the belief (Fig. 3b). Value iteration computes Q values by iteratively applying Bellman updates, $Q_{k+1}(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') V_k(s')$, where $V_k(s) = \max_a Q_k(s, a)$.

Value iteration can be expressed as passing a value image through convolutional and max-pooling layers. The Q function is represented by an $N \times N \times |A|$ image, where each channel corresponds to an action. The first step of a Bellman update is max-pooling along the channels of Q_k which gives a value image V_k. The summation operation in the Bellman update is expressed by an $N \times N$ convolution with $|A|$ filters, f'_T, where the convolutional kernel represents the transition function in M. The output of the convolutional layer is added to a reward image R, which gives Q_{k+1}, the Q image for the next iteration. Rewards depend on the task instance $\boldsymbol{\theta}$, e.g. the goal and obstacles for navigation. We learn a mapping from $\boldsymbol{\theta}$ to the reward image, $R = f_R(s, a|\boldsymbol{\theta})$.

K iterations of Bellman updates are implemented by stacking the convolution and max-pool layers K times with tied kernel weights. After K iterations we get Q_K, the approximate Q values for each state-action pair. From state-action values we get action values by weighing with the belief, $q(a) = \sum_s Q_K(s, a) b(s)$. We obtain the output of the planner through a low-level policy function, f_π, mapping from $q(a)$ defined in M to action outputs defined in $\mathbb{W}$.

POMDP model prior. Finding a suitable neural network structure for a particular problem often involves unintuitive choices in practice. QMDP-net embeds a POMDP model M which provides a principled way to incorporate domain knowledge and choose an appropriate network structure through the components $f_R, f_T, f'_T, f_Z, f_O, f_A, f_B, f_\pi$. We do not need to know much about $\mathbb{W}$ in order to choose a reasonable structure for M. For example, assuming that $\mathbb{W}$ has a local and spatially invariant connectivity structure, we can use convolutions with small kernels in f_T, f_R and f_Z.

In our experiments we choose M based on the structure of the underlying POMDP, $\mathbb{W}$. We use $|S| = N \times N$ for $N \times N$ grid worlds, and A and O that match the size of the action and observation space in $\mathbb{W}$. An exception is the grasping task where the underlying observation space has 64 possible values, while the embedded model has $|O| = 16$. In the maze domain we use $|S| = N \times N \times 4$. We implement f_T, f_R and f_Z using CNN components with 3×3 and 5×5 kernels. f_O is realized by a small fully connected component, f_A is a one-hot encoding function, f_π is a single softmax layer, and f_B is the identity function. We enforce that f_T and f_Z are proper probability distributions by using softmax and sigmoid activations on the convolutional kernels, respectively.

We can adjust the *amount of planning* in QMDP-net by setting K. A large K allows propagating information to more distant states without affecting the number of parameters to learn. However, it results in deeper networks that are computationally expensive to evaluate and more difficult to train.

We used $K = 20 \ldots 116$ depending on the problem size. We were able to transfer policies to larger environments by increasing K up to 450 when executing the policy.

QMDP-net naturally extends to higher dimensional discrete state spaces (e.g. our maze domain) where n-dimensional convolutions can be used [13]. While M is restricted to a discrete space, we can handle continuous tasks through the input and output mappings, f_A, f_O and f_π. In our domains the representation of $\boldsymbol{\theta}$ is isomorphic to the chosen state space S. While this assumption is not required, we rely on it to represent f_T, f_Z, f_R by convolutions with small kernels. Experiments with a more general class of problems is an interesting direction for future work.

5 Experiments

We evaluate QMDP-net in synthetic domains: robot navigation and object grasping. While these tasks are relatively simple in terms of perceptual input, compared with e.g. Atari games, they are challenging, because of the sophisticated long-term reasoning required to deal with partial observability. Since the exact state of the robot is unknown, a successful policy must improve state estimation through sequences of partial and noisy observations and also reason about distant future rewards. Results indicate that QMDP-net policies generalize to new environments better than alternatives, and they also transfer to larger environments. The model learned by QMDP-net can perform better than a “correct" model, given an approximate planning algorithm.

5.1 Setup

Environments are generated randomly. The goal and the underlying initial state are sampled from the free space. The initial belief is chosen uniform over a random fraction of the free space. Expert trajectories are obtained by an approximate QMDP policy. The policy is computed on an underlying POMDP that describes the domain. Note that we do not receive supervision on these POMDPs, and have no access to the underlying states and beliefs along the trajectories. The QMDP policy may fail to reach the goal in same environments, which we exclude from the training set but include in the test set. We briefly describe the domains here. See Appendix C for implementation details.

Grid world navigation. A robot needs to navigate in an unknown building given a floor map and a goal. The robot is uncertain of its own location. It is equipped with a LIDAR that detects obstacles in its direct neighborhood. The world is noisy: the LIDAR may produce false readings, and the robot may fail to execute desired actions (e.g. the wheels slip). We need to obtain a policy without having an accurate model of the sensor and the transition dynamics. We implement a simplified version of this task in discrete $N \times N$ grids (Fig. 1). The robot is given an $N \times N \times 3$ image, $\boldsymbol{\theta}$, that encodes information about the environment: one channel encodes obstacles, another channel encodes the goal, and the last channel encodes the initial belief over states. The robot has 5 actions: moving in the four canonical directions and staying put. LIDAR observations are compressed into four binary values corresponding to obstacles in the four neighboring cells. We consider both a deterministic and a stochastic variant of the task. In the stochastic case the observations are faulty with probability $P_o = 0.1$ independently in each direction; and the robot fails to execute actions with probability $P_t = 0.2$, in which case it stays in place. We train a policy using expert trajectories in $10,000$ random environments, 5 trajectories each. We then test on a separate set of 500 random environments.

Maze navigation. We consider the problem of a differential drive robot finding its way through a maze based on a map, without knowing its own location (Fig. 1d). The task is similar to the grid world navigation in nature, but it is significantly more challenging. First, path finding in a maze requires more computation. Second, the state space now has three dimensions where the third dimension represents four possible orientations of the robot. The robot cannot move in any direction at any time. Instead, it may move forward, turn left, turn right and stay put. The observations are now relative to the robot's orientation which makes localization significantly more difficult (Fig. 4), especially when the initial state is highly uncertain. We train on expert trajectories in $10,000$ randomly generated mazes, and test them in 500 new ones.

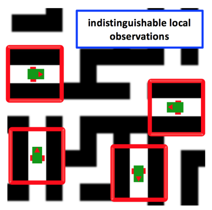

Fig. 4: Observations relative to the robot's orientation are insufficient to determine its state.

Navigation on a large LIDAR map. In this domain we train a navigation policy in synthetic random grids and execute them in significantly larger environments corresponding to real-life buildings (Fig. 1c). We obtain 2-D maps from LIDAR data collected in a set of buildings and processed by SLAM from the Robotics Data Set Repository [11]. The QMDP-net policy is given a preprocessed binary map and is executed in simulation. The simulation employs the same dynamics as in the grid navigation domain based on a classification of obstacles and free space in the LIDAR map.

2-D object grasping. A robot arm picks up novel objects from a table using a two-finger gripper with (noisy) touch sensors mounted on the fingers. The robot is given some information of the target object (e.g. a picture), but it has no visual sensors to observe its location on the table. The robot may use its fingers to perform compliant motions while maintaining contact, as well as to grasp the object. We consider a 2-D variant of this task that has been modeled as a POMDP [12] assuming a feasible grasp point is known. The gripper moves in four directions and has two fingers with 3 touch sensors each. The initial gripper pose is uncertain. We train using expert demonstrations for 20 different objects in 500 randomly sampled configurations each. We then test on 10 previously unseen objects in random configurations. The inputs to the network are readings from the touch sensors, and an image, $\boldsymbol{\theta}$, encoding the environment including the target object, a grasping point, and a belief over the initial gripper position.

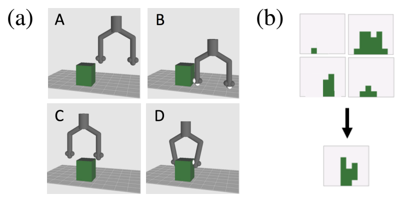

Fig. 5: 2-D grasping task. (a) Illustration borrowed from [3]. (b) Objects from the training set (top) and test set (bottom).

5.2 Network architectures of comparison

For comparison, we gradually relax the structure priors imposed on the network architecture. We create two QMDP-net variants. *Untied QMDP-net* relaxes the constraints on the planning module by untying the weights representing the state-transition function over the different CNN layers. *LSTM QMDP-net* replaces the filter module with a generic LSTM module. We also compare with two architectures that do not embed POMDP structure priors. *CNN+LSTM* is a state-of-the-art deep CNN connected to LSTM. It is similar to the DRQN architecture [9], proposed for reinforcement learning under partially observability. *RNN* is a basic recurrent neural network, with a single fully-connected hidden layer. RNN contains minimum structure specific to planning under partial observability. We train all networks in an imitation learning setting. For fair comparison, we tune the hyperparameters, the number of layers and hidden units, for all networks and use the same set of expert trajectories generated by the QMDP algorithm. Details are available in Appendix C.

5.3 Results and discussion

The main results are reported in Table 1. For each task, we report the success rate and the average number of time steps for task completion. We can compare the completion time meaningfully when the success rates are similar (completion time is calculated only upon successful task completion). Note that rewards are not meaningful in the imitation learning setting.

QMDP-net successfully learns policies that generalize to new environments. When evaluated on new environments, QMDP-net policies have high success rate and fast completion time for almost all tasks, and they perform better than the competitors. We also specifically trained and tested policies in a fixed environment for navigation (Fixed grid in Table 1). Only the initial state and the goal vary. In this case, QMDP-net and alternatives have comparable performance. Even RNN performs very well. Why? In a fixed environment, a network may learn the features of an optimal policy directly without understanding the full environment, e.g., always going towards the goal. To handle different environments, QMDP-net learns how to *plan*, i.e., how to generate an optimal policy.

QMDP-net policies transfer directly to larger domains. It is difficult to learn a policy in a large-scale environment from scratch. A scalable approach would be able to learn a policy in a small environment and "transfer" it to a large environment by repeating the reasoning process. To transfer a learned QMDP-net policy to a much larger environment, we simply expand its planning module by adding more iterations at execution time. Specifically, we trained a policy in randomly generated 30×30 grid worlds with $K = 90$. We then set $K = 450$ and applied the learned policy to 100×101

Table 1: Performance comparison. For each task, we report the success rate in percentage (SR) and the average number of time steps (Time) for task completion. We denote deterministic and stochastic variants of a task by D-n and S-n, where n corresponds to the size of the environment.

	Expert **QMDP**		**QMDP-net**		**Untied QMDP-net**		**LSTM QMDP-net**		**CNN +LSTM**		**RNN**	
Domain	SR	Time	SR	Time	SR	Time	SR	Time	SR	Time	SR	Time
Grid D-10	99.8	8.8	99.6	8.2	98.6	8.3	84.4	12.8	90.0	13.4	87.8	13.4
Grid D-18	99.0	15.5	99.0	14.6	98.8	14.8	43.8	27.9	57.8	33.7	35.8	24.5
Grid D-30	97.6	24.6	98.6	25.0	98.8	23.9	22.2	51.1	19.4	45.2	16.4	39.3
Grid S-18	98.1	23.9	98.8	23.9	95.9	24.0	23.8	55.6	41.4	65.9	34.0	64.1
Maze D-29	63.2	54.1	98.0	56.5	95.4	62.5	9.8	57.2	9.2	41.4	9.8	47.0
Maze S-19	63.1	50.5	93.9	60.4	98.7	57.1	18.9	79.0	19.2	80.8	19.6	82.1
Hallway2	37.3	28.2	82.9	64.4	69.6	104.4	82.8	89.7	77.8	99.5	68.0	108.8
Grasp	98.3	14.6	99.6	18.2	98.9	20.4	91.4	26.4	92.8	22.1	94.1	25.7
Intel lab	90.2	85.4	94.4	107.7	20.0	55.3	-		-		-	
Freiburg	88.4	66.9	93.2	81.1	37.4	51.7	-		-		-	
Fixed grid	98.8	17.4	98.6	17.6	99.8	17.0	97.0	19.7	98.4	19.9	98.0	19.8

and 139×57 real-life environments, represented by LIDAR maps (Fig. 1c). See the result for Intel lab and Freiburg in Table 1. See Appendix A for results on different K settings and additional buildings.

POMDP structure priors are useful for learning complex policies. Moving across Table 1 from left to right, we gradually relax the POMDP structure priors on the network. Untied QMDP-net unties some network parameters. LSTM QMDP-net changes the filter module to a generic LSTM. As the structure priors weaken, so does the overall performance. However, strong priors sometimes over-strain the network and result in degraded performance. For example, we found that tying the weights of f_T and f'_T, the kernels representing T in the filter and the planner, can lead to worse policies. We shed some light on this issue and visualize the learned POMDP model in Appendix B.

QMDP-net learns "incorrect", but useful models. As planning under partial observability is intractable in general, we must rely on approximation algorithms. A QMDP-net encodes both a POMDP model and QMDP, an approximate POMDP algorithm that solves the model. We then train the network end-to-end. This provides the opportunity for learning an "incorrect", but useful model that compensates the limitation of the approximation algorithm, in a way similar to reward shaping in reinforcement learning. Indeed, our results show that QMDP-net achieves higher success rate than QMDP in almost all tasks. In particular, QMDP-net performs well on the well-known Hallway2 domain, which is designed to expose the weakness of QMDP resulting from its myopic planning horizon. The planning algorithm is obviously the same in both QMDP-net and QMDP. However, QMDP-net learns a more effective model from successful expert demonstrations.

6 Conclusion

The QMDP-net is a deep recurrent policy network that embeds POMDP structure priors for planning under partial observability. While generic neural networks learn a direct mapping from inputs to outputs, QMDP-net learns how to *model* and *solve* a planning task. The network is fully differentiable and allows for end-to-end training.

Experiments on several simulated robotic tasks show that learned QMDP-net policies successfully generalize to new environments and transfer to larger environments as well. The POMDP structure prior substantially improves the performance of learned policies. End-to-end training further increases their robustness. In particular, although QMDP-net encodes the QMDP algorithm for planning, learned QMDP-net policies often outperform QMDP.

QMDP is a simple algorithm that makes strong approximations in order to make planning tractable. It would be interesting to explore whether we can embed more sophisticated POMDP algorithms into the network architecture. While these algorithms provide stronger planning performance, their algorithmic sophistication also increases the difficult of learning.

We have so far restricted the work to imitation learning. It would be exciting to extend it to reinforcement learning. Based on earlier work [25, 31], this is indeed promising.

Acknowledgment We thank Leslie Kaelbling and Tomás Lozano-Pérez for insightful discussions that helped to improve our understanding of the problem.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`.

[2] J. A. Bagnell, S. Kakade, A. Y. Ng, and J. G. Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems*, pages 831–838, 2003.

[3] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo. Monte carlo value iteration for continuous-state pomdps. In *Algorithmic foundations of robotics IX*, pages 175–191, 2010.

[4] B. Bakker, V. Zhumatiy, G. Gruener, and J. Schmidhuber. A robot that reinforcement-learns to identify and memorize important previous observations. In *International Conference on Intelligent Robots and Systems*, pages 430–435, 2003.

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

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

[7] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 2017.

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

[9] M. J. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable MDPs. *arXiv preprint*, 2015. URL `http://arxiv.org/abs/1507.06527`.

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

[11] A. Howard and N. Roy. The robotics data set repository (radish), 2003. URL `http://radish.sourceforge.net/`.

[12] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Grasping POMDPs. In *International Conference on Robotics and Automation*, pages 4685–4692, 2007.

[13] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[14] R. Jonschkowski and O. Brock. End-to-end learnable histogram filters. In *Workshop on Deep Learning for Action and Interaction at NIPS*, 2016. URL `http://www.robotics.tu-berlin.de/fileadmin/fg170/Publikationen_pdf/Jonschkowski-16-NIPS-WS.pdf`.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[16] H. Kurniawati, D. Hsu, and W. S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, volume 2008, 2008.

[17] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, pages 362–370, 1995.

[18] M. L. Littman, R. S. Sutton, and S. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems*, pages 1555–1562, 2002.

[19] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[21] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[22] J. Pineau, G. J. Gordon, and S. Thrun. Applying metric-trees to belief-point pomdps. In *Advances in Neural Information Processing Systems*, page None, 2003.

[23] G. Shani, R. I. Brafman, and S. E. Shimony. Model-based online learning of POMDPs. In *European Conference on Machine Learning*, pages 353–364, 2005.

[24] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-agent Systems*, 27(1):1–51, 2013.

[25] T. Shankar, S. K. Dwivedy, and P. Guha. Reinforcement learning via recurrent convolutional neural networks. *arXiv preprint*, 2017. URL `https://arxiv.org/abs/1701.02392`.

[26] D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.

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

[28] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, et al. The predictron: End-to-end learning and planning. *arXiv preprint*, 2016. URL `https://arxiv.org/abs/1612.08810`.

[29] M. T. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[30] C. Stachniss. Robotics 2d-laser dataset. URL `http://www.ipb.uni-bonn.de/datasets/`.

[31] A. Tamar, S. Levine, P. Abbeel, Y. Wu, and G. Thomas. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2146–2154, 2016.

[32] N. Ye, A. Somani, D. Hsu, and W. S. Lee. Despot: Online pomdp planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017.

A Supplementary experiments

A.1 Navigation on a large LIDAR map

We provide results on additional buildings for the LIDAR map navigation domain. LIDAR maps are obtained from [30]. See Section C.4 and Fig. 10 for details. **Intel** corresponds to Intel Research Lab. **Freiburg** corresponds to Freiburg, Building 079. **Belgioioso** corresponds to Belgioioso Castle. **MIT** corresponds to the western wing of the MIT CSAIL building. We note the size of the pre-processed map size, NxM, for each environment. The QMDP-net policy is trained on the $30x30$-D grid navigation domain on randomly generated environments using $K = 90$. Note that alternative networks, except for Untied QMDP-net, are specific to input size and thus they are not applicable.

Table 2: Additional results for navigation on large LIDAR maps.

	Expert **QMDP**		**QMDP-net** K=450		**QMDP-net** K=180		**QMDP-net** K=90		**Untied QMDP-net**	
Domain	SR	Time	SR	Time	SR	Time	SR	Time	SR	Time
Intel 100×101	90.2	85.4	**94.4**	108.0	83.4	89.6	40.8	78.6	20.0	55.3
Freiburg 139×57	88.4	66.9	92.0	91.4	**93.2**	81.1	55.8	68.0	37.4	51.7
Belgioioso 151×35	95.8	63.9	**95.4**	71.8	90.6	62.0	60.0	54.3	41.0	47.7
MIT 41×83	94.4	42.6	91.4	53.8	**96.0**	48.5	86.2	45.4	66.6	41.4

We execute the QMDP-net policy with different K settings, i.e. we add convolutional layers to the planner that share the same kernel weights during execution. In the conventional setting, when value iteration is executed on a fully known MDP, increasing K improves the value function approximation and improves the policy in return for the increased computation. In QMDP-net increasing K has two effects on the overall planning quality. Estimation accuracy of the latent values increases and reward information can propagate to more distant states. On the other hand the learned latent model does not necessarily fit the true underlying model, and it can be overfitted to the K setting during training. Therefore a too high K can degrade the overall performance. We found that $K_{test} = 2K_{train}$ significantly improved success rates in all our test cases. Further increasing $K_{test} = 5K_{train}$ was beneficial in the Intel and Belgioioso environments, but it slightly decreased success rates for the Freiburg and MIT environments.

A.2 Incorrect model gives better QMDP policy

We demonstrate how an “incorrect" model can result in better policies using the approximate QMDP algorithm. We compute QMDP policies on a POMDP with modified reward values, then we evaluate the obtained policies using the original rewards. We use the deterministic 29×29 variant of the maze domain where QMDP did poorly. The motivation for choosing different rewards is to break symmetry in the model and to implicitly encourage information gathering and compensate for the one-step look-ahead approximation in QMDP. Finding appropriate reward values manually is unintuitive and difficult. **Modified 1.** We increase the cost for the stay actions to 20 times of its original value. **Modified 2.** We increase the cost for the stay actions to 50 times of its original value; and the cost for the turn right action to 10 times of their original values.

Table 3: QMDP policies computed on an “incorrect” model and evaluated on the ”correct" model.

Variant	SR	Time	Original reward
Original	63.2	54.1	1.09
Modified 1	65.0	58.1	1.71
Modified 2	**93.0**	71.4	4.96

Why does the “correct" model give poor policies? At a given point the Q value for a set of possible states may be high for the turn left action and low for the turn right action; while for another set of states it may be the opposite way around. In expectation, both next states have lower value than the current one, thus the policy chooses the stay action, the robot does not gather information and it gets stuck. Results demonstrate that planning on an “incorrect” model may improve the performance on the “correct” model.

B Visualizing the learned model

Value function. First we visualize the value function predicted by QMDP-net on the 18×18 stochastic grid navigation domain using $K = 54$. States close to the goal have high values.

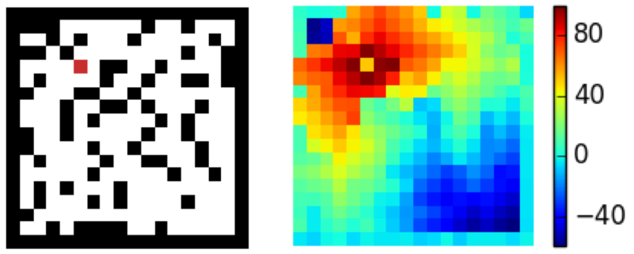

Fig. 6: Environment $\boldsymbol{\theta}$ (without the initial belief) and the predicted value image V_K.

Belief propagation. We then evaluate the execution and the propagation of the belief. The first row in Fig. 7 shows the environment including the goal (red) and the unobserved state of the robot (blue). The second row shows ground-truth beliefs. The third row shows beliefs predicted by QMDP-net. The last row shows the difference between the ground-truth and predicted beliefs. Note that we do not receive supervision on the ground-truth beliefs for training QMDP-net except for the initial belief.

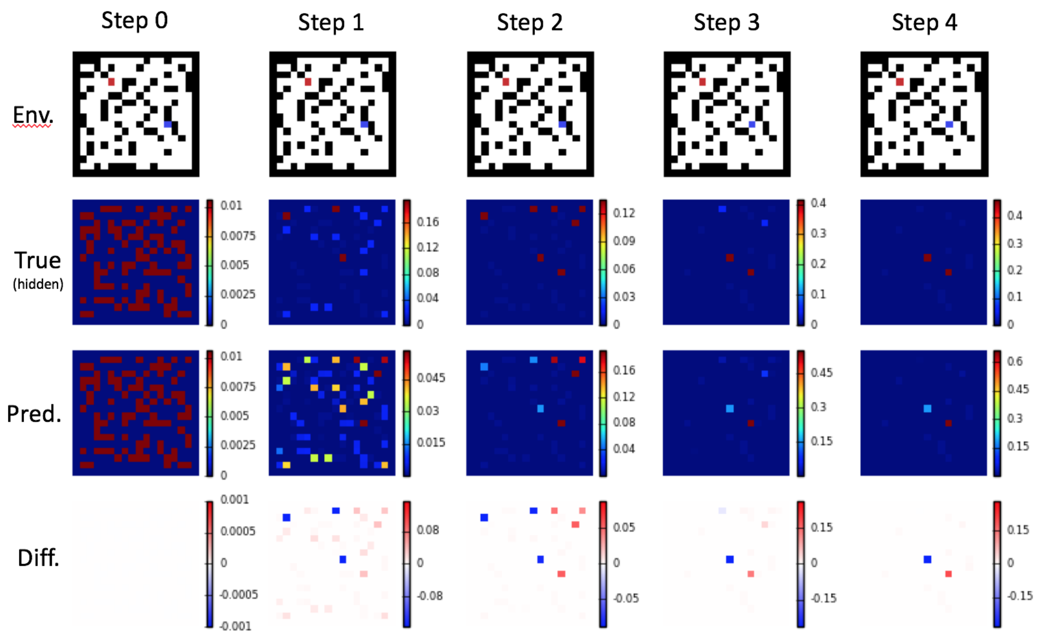

Fig. 7: Policy execution and belief propagation in the 18×18 stochastic grid navigation domain.

The figure demonstrates that QMDP-net was able to learn a reasonable filter for state estimation even in a noisy environment. In the depicted example the initial belief is uniform over approximately half of the state space (Step 0). Due to the highly uncertain initial belief and the noisy observation the robot stays in place for two steps (Step 1 and 2). After two steps the state estimation is still highly uncertain, but it is mostly spread out right from the goal. Therefore, moving left is a reasonable choice (Step 3). After an additional stay action (Step 4) the belief distribution is small enough and the policy starts moving towards the goal (not shown).

Transition function. Next, we plot the learned and ground-truth transition functions. The first row shows the ground truth transition function in the underlying POMDP. The second row shows f_T, the transition model in the filter; and the third row shows f'_T, the transition model in the planner.

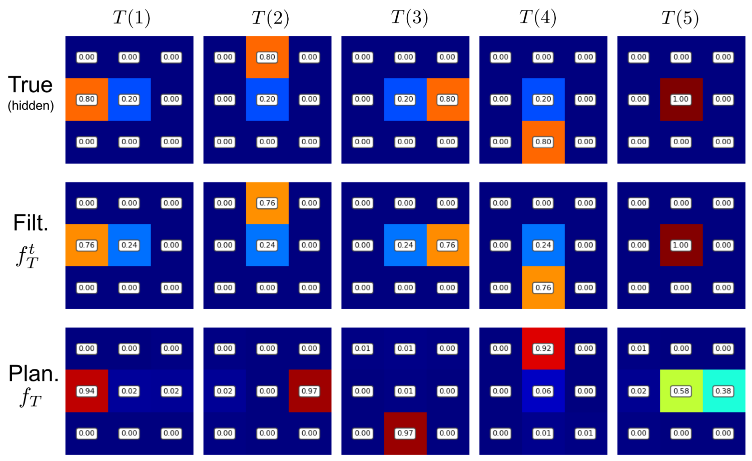

Fig. 8: Learned transition function in the 18×18 stochastic grid navigation domain.

Reward function. Next we show the learned reward image for each action in the embedded POMDP. While these images do not directly correspond to rewards in the underlying POMDP, they are reasonable: obstacles are assigned negative rewards and the goal is assigned a positive reward. Note that the learned reward images should be interpreted together with the transition model, as they correspond to rewards after taking an action.

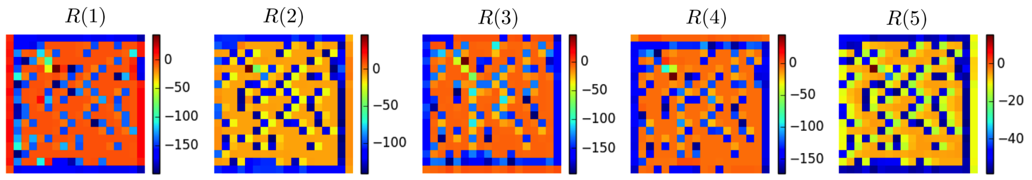

Fig. 9: Learned reward images in the 18×18 stochastic grid navigation domain.

C Implementation details

C.1 Grid navigation

We implement the grid navigation task in discrete $N \times N$ grids. The robot has 5 actions: moving in the four canonical directions and staying put. Observations are four binary values corresponding to obstacles in the four neighboring cells. We are given an $N \times N \times 3$ image that encodes information about the environment. The first channel encodes obstacles, 1 for obstacles, 0 for free space. The second channel encodes the goal, 1 for the goal, 0 otherwise. The third channel encodes the initial belief over robot states, each pixel value corresponds to the probability of the robot being in the state. The robot receives a reward of -0.1 for each step, $+20$ for reaching the goal, and -10 for bumping into an obstacle. In the stochastic variant of the problem (denoted by -S) the observations are faulty with probability $P_o = 0.1$ independently in each direction. Since we receive observations from 4 direction the probability of receiving a correct observation vector is $0.9^4 = 0.656$. The robot also fails to execute actions with probability $P_t = 0.2$, in which case it stays in place.

We use $10,000$ random grids for training, where each cell has $p = 0.25$ probability of being an obstacle. Initial and goal states are sampled from the free space uniformly. We exclude samples where there is no feasible path. The initial belief is uniform over a random fraction of the free space which includes the underlying initial state. More specifically, the number of non-zero values in the initial-belief are sampled from $\{1, 2, \ldots N_f/2, N_f\}$ where N_f is the number of free cells for the particular environment. In each environment we generate 5 expert trajectories using an approximate QMDP policy computed on the underlying POMDP defined above. Training samples consist of an image encoding the environment and a sequence of actions and observations along the expert trajectory. Note that we do not access the true beliefs after the first step nor the underlying states along the trajectory. We also receive no supervision on the underlying POMDP model (rewards, transition and observation function); however, we leverage some knowledge on the structure of the POMDP to choose the structure of the model embedded in QMDP-net (size of state space, action space and observation space).

We test on a set of 500 environments generated separately in equal conditions. We declare failure after $10N$ steps without reaching the goal. Note that in some cases the approximate expert policy may fail to reach the goal. We exclude these samples from the training set but include them in the test set.

We use a POMDP model prior with state, action and observation spaces that match the underlying POMDP. The transition function in the filter f_T and the planner f'_T are both 3×3 convolutions, but their weights are not tied. We apply a softmax function on the kernel matrix so its values sum to one. The reward image mapping, f_R is a CNN with two convolutional layers. The first has a 3×3 kernel, 150 filters, ReLU activation. The second has 1×1 kernel, 5 filters and linear activation. The observation model, f_Z, is a similar 2 layer CNN. The first convolution has a 3×3 kernel, 150 filters, linear activation. The second has 1×1 kernel, 17 filters and linear activation. The action mapping, f_A, is a one-hot encoding function. The observation mapping, f_O, is fully connected network with one hidden layer (17 units, tanh), 17 output units and softmax activation. The low-level policy function, f_π, is a single softmax layer. The state space mapping function, f_B, is an identity function. Finally, we choose the number of iterations in the planner module, $K = \{30, 54, 90\}$ for grids of size $N = \{10, 18, 30\}$ respectively.

The 3×3 convolutions in f_T and f_Z imply that T and O are spatially invariant and local. In the underlying problem the locality assumption holds but spatial invariance does not: transitions depend on the arrangement of obstacles. Nevertheless, the additional flexibility in the model allowed QMDP-net to learn high-quality policies, e.g. by shaping the rewards and the observation model.

C.2 Maze navigation

In the maze domain a differential drive robot has to navigate to a given goal. We generate random mazes on $N \times N$ grids using Kruskal's algorithm. The state space has 3 dimensions where the third dimension represents 4 possible orientations of the robot. The goal configuration is invariant of the orientation. The robot now has 4 actions: move forward, turn left, turn right and stay put. The initial belief is chosen in a similar manner but in the 3-D space. The observations are identical to grid navigation case but they are relative to the robot's orientation, which significantly increases the difficulty of state estimation. The stochastic variant (denoted by -S) has a motion and observation noisy identical to the grid navigation domain. Training and test data is prepared identically to the grid navigation domain. We use $K = \{76, 116\}$ for mazes of size $N = \{19, 29\}$ respectively.

We use POMDP prior in QMDP-net with a 3-dimensional state space of size $N \times N \times 4$ and an action space with 4 actions. The components of the network are chosen identically to the previous case, except that all CNN components operate on 3-D images of size $N \times N \times 4$. We treat the third dimension as channels of the input image and use 2-D CNNs: if the output of the last convolutional layer is of size $N \times N \times N_c$ in the 2-D variant, it is of size $N \times N \times 4N_c$ in the 3-D case. When necessary, these images are transformed into a 4 dimensional form $N \times N \times 4 \times N_c$ and the max-pool or softmax activation is computed along the last dimension.

C.3 Object grasping

We consider a 2-D implementation of the grasping task based on the POMDP model proposed by Hsiao et al. [12]. The object and the gripper are represented in a discrete grid. The gripper moves in four directions and has two fingers with 3 touch sensors each. Hsiao et al. [12] concentrated on the difficulties of planning involved with high uncertainty and solved manually designed POMDP representation for single objects. We phrase the problem as a learning task where we have no access to a model and we do not know all objects in advance. In our setting the robot receives an image of the target object and a feasible grasp point, but it does not know its relative position to the object.

We represent the work space by a 14×14 grid, and the gripper by a U shape in the grid. We have 30 objects of different sizes up to 6×6 grid cells. Each object has at least one cell on its top that the gripper can grasp. In each trial the object is placed on the bottom of the work space at a random location. The 6 touch sensors on the fingers indicate contact with the object or reaching the limits of the work space. The initial gripper pose is unknown; the belief over possible states is uniform over a random fraction of the upper half of the work space. The touch sensors produce an incorrect reading independently with probability 0.1; and the gripper fails to move with probability 0.2. The robot receives a reward of 1 for reaching the grasp point and 0 for every other state. The local observations, o_i, are readings from the touch sensors. The task parameter $\boldsymbol{\theta}$ is an images with three channels. The first channel encodes the environment with an object; the second channel encodes the position of the target grasping point; the third channel encodes the initial belief over the gripper position.

We use a POMDP prior with $|S| = 14 \times 14$, $|A| = 4$ and $|O| = 16$. Note that the underlying POMDP has 64 observations. The network components are chosen similarly to the grid navigation case; however the first convolution kernel in f_Z is increased to 5×5. We set the number of iterations $K = 20$.

For training, we generate 500 expert trajectories by QMDP on the true model for 20 of the objects each. We test the learned policies on 10 new objects, that have not been seen during training, in 20 random scenarios each.

C.4 Navigation on a large LIDAR map

We obtain real-world building layouts using 2-D laser data from the Robotics Data Set Repository [11]. More specifically, we use SLAM maps preprocessed to gray-scale images available at [30]. We downscale the raw images to NxM and classify each pixel to be free or an obstacle by simple thresholding (see Fig. 10). We obtain a ground-truth POMDP model with identical parameters to the grid navigation domain where the obstacles are defined by the preprocessed map. The initial state and initial belief are chosen identically to the grid navigation case. The QMDP-net policy is trained on the $30x30$-D grid navigation domain on randomly generated environments using $K = 90$. It is then executed on the real-world map using $K = 450$.

C.5 Hallway

The Hallway2 navigation problem was proposed by Littman et al. [17] and has been used as a benchmark problem for POMDP planning [24]. It was specifically designed to expose the weakness of QMDP resulting from its myopic planning horizon. Nevertheless, QMDP-net was able to learn a model that is significantly more effective using the same QMDP algorithm. Hallway2 is a particular

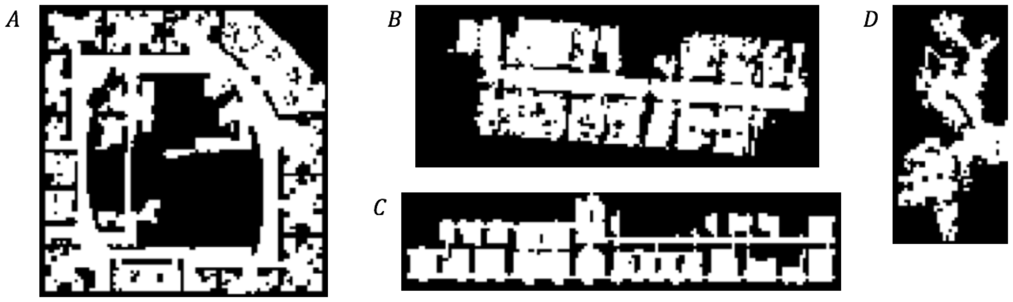

Fig. 10: Preprocessed $N \times M$ maps. **A,** Intel Research Lab, 100×101. **B,** Freiburg, building 079, 139×57. **C,** Belgioioso Castle, 151×35. **D,** western wing of the MIT CSAIL building, 41×83.

instance of the maze problem that involves more complex dynamics and high noise. For details we refer to the original problem definition [17].

We train a QMDP-net on random 8×8 grids generated similarly to the grid navigation case, but using transitions that match the original Hallway2 POMDP. We then execute the learned policy on a particularly difficult instance of this problem that embeds the Hallway2 layout in an 8×8 grid. The initial state is uniform over the full state space. In each trial the robot starts from a random underlying state. The trial is deemed unsuccessful after 251 steps.

This POMDP model is difficult to solve for QMDP due to the relatively complex and noisy dynamics.

C.6 Training technique

We train all networks, QMDP-net and alternatives, in an imitation learning setting. The loss is defined as the cross-entropy between predicted and demonstrated actions along the expert trajectories. There is no supervision on the underlying POMDP model.

The networks are implemented in Tensorflow [1]. We train using backpropagation through time on mini-batches of 100. We use RMSProp optimizer with 0.9 decay rate and 0 momentum setting. The learning rate was set to 1×10^{-3} for QMDP-net and 1×10^{-4} for the baseline networks. We limit the number of backpropagation steps to 4 for QMDP-net and its untied variant; and to 6 for the other alternatives, which gave slightly better results. We used a combination of early stopping with patience and exponential learning rate decay of 0.9. In particular, we started to decrease the learning rate if the prediction error did not decrease for 30 consecutive epochs on a validation set, 10% of the training data. We performed 15 iterations of learning rate decay.

In our partially observable domains predictions are increasingly difficult along a trajectory, as they require multiple steps of filtering, i.e. integrating information from a long sequence of observations. Initially we limit the number of steps along the expert trajectories when training our network as well as the baseline architectures. In particular, we perform two rounds of the training method described above, in each round using a different limit for the number of steps along the expert trajectories, L_r for round r. For training QMDP-net and its untied variant we used $L_1 = 4$ and $L_2 = 100$. For training the other baselines we used $L_1 = 6$ and $L_2 = 100$, which gave better results. On the fixed grid variant we found that a low L_r setting degrades the final performance of the generic baselines, therefore we used a single training round with $L_1 = 100$.

C.7 Architectures for comparison

We compare QMDP-net to two of its variants where we remove some of the POMDP priors embedded in the network (Untied QMDP-net, LSTM QMDP-net). We also compare to two generic architectures that do not embed structural priors for decision making (CNN+LSTM, RNN).

We obtain Untied QMDP-net by untying the kernel weights in the convolutional layers that implement value iteration in the planner module. We also remove the softmax activation on the kernel weights. This is equivalent to allowing a different transition model at each iteration of value iteration, and allowing transition probabilities that do not sum to one. In principle Untied QMDP-net can represent the same function as QMDP-net and it has some additional flexibility. However, Untied QMDP-net has more parameters to train as K increases. Having more parameters make training more difficult, especially on more complex domains or small amount of training data.

In the LSTM QMDP-net we replace the filter module of QMDP-net with a generic LSTM network but keep the value iteration implementation in the planner. The output of the LSTM component is a belief estimate which is input to the planner module of QMDP-net. We first process the channels of the parameter image $\boldsymbol{\theta}$, that encode the environment and goal, by a CNN. We separately process the action-observation input vector with a two-layer fully connected component. These processed inputs are concatenated into a single vector which is the input of the LSTM layer. The size of the LSTM hidden state and output is chosen to match the number of states for the domain, e.g. N^2. We initialize the hidden state of the LSTM using the appropriate channel of the $\boldsymbol{\theta}$ image that encodes the initial belief.

The CNN+LSTM is a state-of-the-art deep convolutional network with LSTM cells. It is similar in structure to DRQN [9], which was used for learning partially observable Atari games in a reinforcement learning setting. Note that we train the networks in an imitation learning setting using the same set of expert trajectories, and not using reinforcement learning, so the comparison is fair. The CNN+LSTM network has more structure to encode a decision making policy compared to a vanilla RNN, and it is also more tailored to our input representation. We process the image input by a CNN component and the vector input by a fully connected network component. These are then combined into a single vector and input to an LSTM layer.

The considered RNN architecture is a vanilla recurrent neural network with 512 hidden units and tanh activation. At each step inputs are transformed into a single concatenated vector. The outputs are obtained by a fully connected layer with softmax activation.

We performed hyperparameter search on the number of layers and hidden units for both alternative networks. In particular, we ran trials in the deterministic grid navigation domain and for each architecture we chose the best parameterization found. We applied the same settings for other domains. We adjusted training parameters (learning rate, batch size) on this domain as well.

We also considered alternative architectures for comparison, including GRU and ConvLSTM network. The latter is a variant of LSTM where the fully connected layers are replaced by convolutions. These architectures performed worse than CNN+LSTM in most of our domains.