Locally-Connected Interrelated Network: A Forward Propagation Primitive

Nicholas Collins[1] and Hanna Kurniawati[2]

[1] School of Information Technology and Electrical Engineering
University of Queensland, Brisbane, QLD, Australia
nicholas.collins2@uq.net.au,
[2] Research School of Computer Science
Australian National University, Canberra, ACT, Australia
hanna.kurniawati@anu.edu.au

Abstract. End-to-end learning for planning is a promising approach for finding good robot strategies in situations where the state transition, observation, and reward functions are initially unknown. Many neural network architectures for this approach have shown positive results. Across these networks, seemingly small components have been used repeatedly in different architectures, which means improving the efficiency of these components has great potential to improve the overall performance of the network. This paper aims to improve one such component: The forward propagation module. In particular, we propose Locally-Connected Interrelated Network (LCI-Net) —a novel type of locally connected layer with unshared but interrelated weights— to improve the efficiency of information propagation and learning stochastic transition models for planning. LCI-Net is a small differentiable neural network module that can be plugged into various existing architectures. For evaluation purposes, we apply LCI-Net to QMDP-Net; QMDP-Net is a neural network for solving POMDP problems whose transition, observation, and reward functions are learned. Simulation tests on benchmark problems involving 2D and 3D navigation and grasping indicate promising results: Changing only the forward propagation module alone with LCI-Net improves QMDP-Net generalization capability by a factor of up to 10.

1 Introduction

Stochastic planning requires stochastic models of the state transition, observations, and an objective function. However, such models are not always available, and attaining them can be difficult, especially when the dynamics of both the robot and the environment must be accounted for. To overcome this difficulty, end-to-end deep learning based approaches for combined planning and learning have been proposed and have shown promising results. Many architectures have been proposed[2,3,9,12,16,17,21,22]. These different architectures often share common components and some of these components even appear repeatedly within a single network. Such components are akin to "primitives" in planning, and therefore, we hypothesise that improving the efficiency of such components could substantially improve the capability of neural-network based combined planning and learning. This paper focuses on improving the efficiency of one such component: the Forward propagation module — that is, the neural network component that propagates information on the basis of learned stochastic models of the state transition function.

A straightforward implementation of the forward propagation module often requires many parameters to be learned. Therefore, to reduce training time and data requirement, existing architectures simplify the models being learned. One commonly used approach is to transform states to abstract states and learn the transition models with respect to these abstract states, rather than states (e.g., [2,16,22]). This is generally done via an auto-encoder: An encoder simplifies the current state into an abstract state, predicts the next abstract state via a fully connected layer, and then decodes the subsequent abstract state back into the original state. Transition from one abstract state to the next indeed requires a much smaller number of parameters to be learned. However, one needs to also learn the parameters for the auto-encoder. Moreover, the auto-encoder needs to be deep enough to generate a sufficiently small feature space, which generally increases the number of parameters to be learned.

Other architectures learn a state transition function with respect to the entire state space, but reduce the parameters to be learned by exploiting the fact that state transition are generally local, and by assuming that they depend on actions, rather than state-action pairs (e.g., [3,9,12,17,21]). This is done via weight sharing in a convolution network, where the transition function for each action is represented as a kernel, whose size is much smaller than the size of the state space. This architecture substantially reduces the number of parameters to be learned but, is more constrained in its function representation.

To take the best of both worlds, in this paper, we propose a forward propagation module, called Locally-Connected Interrelated Network (LCI-Net). Key to LCI-Net is a novel locally-connected network layer with indirectly interrelated weights. It enables LCI-Net to exploit the locality property present in most transition functions and applies the learned transition to the original states, rather than abstract states, while learning a transition function that depends on pairs of abstract states and actions. LCI-Net is differentiable and is designed for multi-task learning, in the sense that LCI-Net learns a stochastic model of system's dynamics for a multitude of scenarios at once.

LCI-Net is a simple module that can be plugged into various neural-network architectures that require information propagation governed by learned transition functions, such as model-based reinforcement learning and Bayesian filtering. In this paper, we evaluate LCI-Net by applying it to the QMDP-Net architecture[9] —a neural network architecture that finds policies for Partially Observable Markov Decision Processes (POMDPs) problems whose transition, observation, and reward functions are initially unknown and are learned from data in an end-to-end fashion. In particular, we use LCI-Net to replace QMDP-Net's Forward Propagation module, both in its planning and Bayesian filter modules. Since QMDP-Net's Bayesian filter is the End-to-End learnable Histogram Filter (E2E-HF)[7], our evaluation applies LCI-Net to E2E-HF architecture too. We evaluate the performance of LCI-Net on various 2D and 3D navigation and grasping benchmarks, and evaluate the results of learning on problems of the same class but with much larger state and observation spaces. Simulation results indicate that replacing only the forward propagation component of QMDP-Net with LCI-Net improves its generalization capability by a factor of up to 10.

2 Background and Related Work

2.1 Background

Although LCI-Net can be applied to various neural-network architectures that require a forward propagation module, to make the explanation concrete, in this paper, we focus on applying LCI-Net to compute a good POMDP policy when the POMDP model is not known a priori.

Formally, a POMDP[8,20] is described by an 8-tuple $\langle S, A, O, T, Z, R, \gamma \rangle$, where S is the set of *states*, A is the set of *actions*, and O is the set of *observations*. At each step, the agent is in some hidden state $s \in S$, takes an action $a \in A$, and moves from s to another state $s' \in S$ according to a conditional probability distribution $T(s, a, s') = P(s'|s, a)$, called the transition probability. The current state s' is then partially revealed via an observation o drawn from a conditional probability distribution $Z(s', a, o) = P(o|s', a)$ that represents uncertainty in sensing. After each step, the agent receives a reward $R(s, a)$, if it takes action a from state s. Due to uncertainty in both the effects of actions and observations perceived, a POMDP agent never knows its exact state, and represents this uncertainty as distributions over states, called beliefs, and denoted as $b \in B$. The solution to a POMDP problem is then a mapping from beliefs to actions, called policy π, that maximises the expected total reward, i.e.,

$$V^*(b) = \max_{a \in A} \left[\sum_{s \in S} R(s, a)\ b(s) + \gamma \sum_{o \in O} P(o|b, a)\ V^*(\tau(b, a, o)) \right] \tag{1}$$

where $\gamma \in (0, 1)$ is a discount factor to ensure the optimisation is well defined, and $\tau(b, a, o)$ is the belief after action $a \in A$ is applied to b and observation $o \in O$ is perceived, computed as:

$$\tau(b, a, o)(s') = \eta Z(s', a, o) \sum_{s \in S} T(s, a, s') b(s) \quad ; \quad \eta \text{ is a normalisation factor} \tag{2}$$

When the transition, observation, and/or reward functions are a priori unknown, finding a POMDP policy can be formulated as reinforcement learning or imitation learning problems. LCI-Net can be applied to replace the forward propagation modules of methods in both learning techniques. In this paper, we apply LCI-Net to imitation learning.

2.2 Related Work

Recently, there has been a growing body of works that apply deep learning for model free learning to solve large scale POMDPs when the model is not fully known. For instance, [5] implemented a variation of DQN [15] which replaces the final fully connected layer with a recurrent LSTM layer to solve partially observable variants of Atari games. The work in [14] applied convolutional neural networks with multiple recurrent layers for the task of navigating within a partially observable maze environment. The learned policy is able to generalise to different goal positions within the learned maze, but not to previously unseen maze environments.

More recently, success has been achieved with methods that embed specific computational structures representing a model and algorithm within a neural network and training the network end-to-end, a hybrid approach which has the potential to combine

the benefits of both model-based and model-free methods. For instance, [21] developed a differentiable approximation of value iteration embedded within a convolutional neural network to solve fully observable Markov Decision Process (MDP) problems in discrete space, while [18] implemented a network with specific embedded computational structures to address the problem of path integral optimal control with continuous state and action spaces. These works focus only on cases where the state is fully observable.

By combining the ideas in the above work with recent work on embedding Bayesian filters in deep neural networks[4,7,10], one can develop neural network architectures that combine model-free learning and model-based planning for POMDPs. For instance, [19] implemented a network which implements an approximate POMDP algorithm based on Q_{MDP} [13] by combining an embedded value iteration module with an embedded Bayesian filter. Modules are trained separately, with a focus on learning transition and reward models over directly learning a policy. More recently, [9] developed QMDP-Net, which implements a Q_{MDP} approximate POMDP algorithm to predict approximately optimal policies for tasks in a parameterised domain of environments. Policies are learned end-to-end, focusing on learning an "incorrect but useful" model which learns to optimise policy performance over model accuracy.

Recently, deep learning has been viewed as a new programming paradigm, called differentiable programming, where algorithms and data are implemented as differentiable neural network blocks[1,11]. Neural network "primitives", such as a forward propagation module, can be viewed as one of the programming blocks, albeit one focused for stochastic planning. In this paper, we focus on improving the efficiency of this particular module.

3 LCI-Net

LCI-Net is a forward propagation module that can be embedded into various neural network architectures that combine planning and model learning, such as [2,3,9,12,17,21,22]. Key to LCI-Net are *locally-connected layers with indirectly interrelated weights*. This new type of layer enables efficient information propagation governed by transition models that are more expressive than possible with standard spatial convolution layers, which in turn improves the overall performance and generalisation capability.

LCI-Net is suitable for multi-task learning in partially observable stochastic environments. To show this capability of LCI-Net, we focus on the problem of learning a near optimal policy, end-to-end, for acting in a parameterized set of partially observable scenarios: $\mathcal{W}_\Theta = \{W(\theta)|\theta \in \Theta\}$, where Θ is the set of all possible parameter values. Each parameter θ describes properties of the scenarios such as obstacle geometry and materials, position of static and dynamic obstacles, goal location, and initial belief distribution for a given task and environment. Moreover, the problems of deciding how to act in the various scenarios in $\mathcal{W}_\Theta$ are defined as POMDPs with a common state space $\underline{S}$, action space $\underline{A}$ and observation space $\underline{O}$ but without a priori known transition, action, and observation functions.

We assume the neural network for solving the above problem embeds an internal model $M(\theta) = \langle S, A, O, f_T(.|\theta), f_Z(.|\theta), f_R(.|\theta)\rangle$, where S, A and O are the state space, actions space, and observation space respectively, which are manually specified, and constant across the set of tasks in the task domain. The notations $f_T(.|\theta)$,

$f_Z(.|\theta)$, and $f_R(.|\theta)$ are the transition function, observation function, and reward function, respectively, and are represented by sub-networks which are trained end-to-end to maximise policy performance. Note that $M(\theta)$ does not necessarily represent the "true" underlying model. End-to-end learning enables the learned model to be "incorrect but useful", differing from the "true" model but producing near optimal policies when combined with the embedded solver in situations where applying the solver directly to the "true" model would not yield an optimum policy due to the solver limitations, such as when Q_{MDP} heuristic is used to approximate POMDP solutions.

LCI-Net is a neural network module that learns and propagates information via the transition function $f_T(.|\theta)$. It consists of two inter-related structures. The first structure represents $f_T(.|\theta)$. Key in this structure is a representation suitable for robust learning, in the sense that the results of learning transfer across different tasks in the task domain and generalise beyond the training set. The second structure is designed to efficiently propagate information, such as state-action values and beliefs, through space over a single time step, in accordance to the transition function. Key to this is the new locally-connected layer with indirectly interrelated weights that allows the two structures to merge seamlessly. The following subsections describe these two structures in details.

3.1 Representing Transition Function

The transition function is parameterized by the current state $s \in S$, the action $a \in A$, and the subsequent state $s' \in S$. Therefore, a straightforward modelling of such a function requires $|S|^2|A|$ parameters to be learned, resulting in very large transition models for problems beyond trivial size. To reduce this learning complexity without sacrificing robustness, known information about the problem's structure should be encoded in the neural network representation of the transition function.

To that end, spatial convolution layers have often been used. This is not surprising because convolution layers encode the constraints of locality and positional invariance: The value of an output is influenced only by inputs within some local neighbourhood of the position of the output, and is independent of the position of the output. This locality assumption is suitable for transition functions. In general, only local environment influences the effect of actions within a single time-step. For instance, only surrounding obstacles (within the maximum distance an agent can traverse) matter to determine whether collision will occur within a single time-step. However, the position invariance assumption is rather problematic. This invariance assumes that the local information in all parts of the state space are the same, which in general is false.

Such an invariant means the learned transition model cannot represent important characteristics, such as the dynamic differences between a mobile robot moving in free space and moving near an obstacle, and between a robotic grasper applying force inside and outside the friction cone of a surface. In end-to-end learning, the reduced expressiveness of the learned transition function can be compensated for by the learned reward function. However, it comes at a cost, as we will see in Section 5.2. LCI-Net proposes an alternative to spatial convolution layers that aims to relax the restriction on positional invariance while encoding an additional form of locality. Fig. 1 illustrates this structure.

Specifically, LCI-Net formulates a representation of the transition model in the form $T(s, a, \delta s)$, where $s \in S$, $a \in A$, and δs is the relative change to s after action a

is performed. The transition probabilities for any state s depend only on $h(s)$, the local features of s, and $T(s_1, a, \delta s) = T(s_2, a, \delta s)$ when $h(s_1) = h(s_2)$. Under this formulation, the transition probability is learned for each combination of state local features, action, and relative change in state. The support of T in each dimension of S is constrained to lie within r distance, where r is the maximum distance the agent can move in a single dimension of S within one time-step. Similar to existing approaches, a spatial convolution layer network, with kernel width $2r + 1$, is used to learn $f_T(.|\theta)$. However, unlike existing approaches in which the learned independent kernel weights represent transition probabilities, LCI-Net learns a set of kernel weights to predict the transition probabilities for each state–action pair, based on local environment features. These learned weights are shared, enabling information learned about the transition probabilities for one state to generalise to other states. Moreover, the number of learned weights can be made independent of the size of the state space, allowing a transition model trained on a set of small environments to be applied in larger environments for evaluation, thereby improving scalability and generalisation capability.

End-to-end learning depends on differentiable approximations of planning and state estimation algorithms to allow error signals to be propagated backwards throughout all stages of the network for training. When the transition model component is formulated as $T(a, \delta s)$, as in prior works, the effect of transition can be represented by standard spatial convolution, where the learned transition probabilities form the convolution kernel. This is not possible with the more expressive formulation $T(s, a, \delta s)$, as different weights must be applied in different parts of the state space image. Applying this transition model within an embedded algorithm requires a new type of differentiable structure. To address this requirement, we introduce the concept of *locally-connected layers with indirectly interrelated weights* for information propagation.

3.2 Propagating Information

To enable information such as state-action values or beliefs to be propagated based on a set of transition probabilities in the form $T(s, a, \delta s)$, LCI-Net constructs *locally-connected layers with indirectly interrelated weights* (illustrated in Fig. 1).

Let $X_t(s)$ be some function defined over the state space at time step t and state $s \in S$. In this work, $X_t(s)$ may represent a set of state values $V_t(s)$ or a belief distribution $b_t(s)$. The transition operator takes $X_t(s)$ as input. This input is duplicated $|A|$ times and stacked along a new axis to create channels corresponding to each possible action.

For each channel (aka. each action in A), one shift operation is applied to the image for each direction $\delta s \in D$, where D is a set of directions (relative changes in position). This set may be selected either as the complete set of all possible relative changes in position within some fixed distance, or be some restricted subset, e.g. North, South, East and West in 2D space. The type of set of shifts is a hyper-parameter for the network. The resulting shifted images are stacked along an additional new axis. Let this tensor be called $\bar{X}(s, a, \delta s)$. In the case where $|S|$ has 2 dimensions and size $n \times m$, the shape of $\bar{X}$ is $(n, m, |A|, |D|)$. $\bar{X}(s, a, \delta s)$ is multiplied by the transition model $T(s, a, \delta s)$ predicted by the transition model component network $f_T(.|\theta)$, which applies the effect of the transition probabilities, weighting the value at each shift direction by the probability of a transition in that direction occurring. A sum is then performed over the $|D|$

axis of the tensor to produce an expectation of the future value after transition dynamics are applied.

This network structure can be viewed as analogous to a locally connected network layer (in which locality is encoded, but weights are not shared), where the weights applied in each weighted sum are not independent trainable variables, but are instead provided by an external tensor, $T(s, a, \delta s)$. This means that while the weights are not directly shared, they remain interrelated in that they are produced by $f_T(.|\theta)$, and so are each related to their local environment by the same weights (the kernel of $f_T(.|\theta)$).

This new type of locally-connected layer with indirectly interrelated weights structure provides a compromise which combines the superior generalisation capability of spatial convolution layers with the greater expressiveness of locally connected layers, while encoding algorithmic priors that are well suited to the problems of planning and state estimation. In the next subsection, we will elaborate the efficiency of LCI-Net in terms of the number of learned parameters.

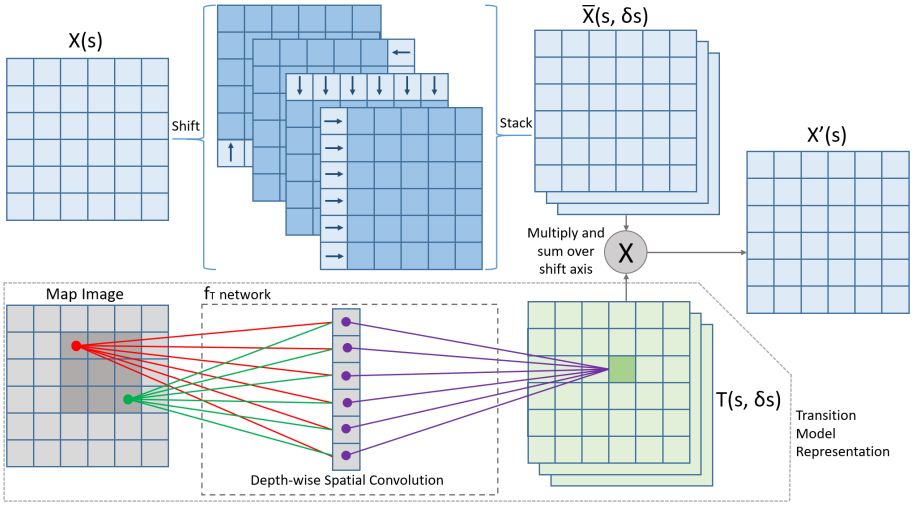

Fig. 1. Transition Operator based on Locally Connected Layer with Indirectly Interrelated Weights, shown with 4 shift operations, simplified to show propagation for only one action rather than all actions in the action space.

3.3 Complexity Analysis

To enable scaling to real world applications, it is essential that efficient time and space complexity are maintained. Compared to prior methods that use spatial convolution to learn the transition probabilities, LCI-Net introduces only a small increase in the number of trainable variables and number of operations required, and has the same asymptotic complexity in terms of state and action space size.

Let r be the maximum range within which the agent can move in one time step. Equivalently, r is the maximum distance within which an environment feature such as an obstacle can influence the movement of the agent. This range corresponds to a

convolution kernel width of $k = 2r + 1$. This kernel width applies to both the transition dynamics convolution in prior methods, and the $f_T(.|\theta)$ network in LCI-Net.

In LCI-Net, the number of channels is equal to the product of the size of the action space $|A|$ and the number of shift directions, $|D|$, giving a total number of learnable parameters as $k^2|A||D|$. As a comparison, the number of channels in prior methods is equal to the number of available actions, which gives a total of $k^2|A|$ parameters to be learned. Furthermore, no additional learnable parameters are introduced outside of the $f_T(.|\theta)$ component network. When D is selected to be the set of all possible shifts within the maximum range r, and the number of dimensions of S is fixed, the size of D is independent of the size of the number of states in S. This gives linear complexity in terms of the number of actions, and constant complexity in terms of the number of states, equal to that of prior methods.

In terms of number of operations required, LCI-Net requires one multiplication and one addition for each combination of state, action, shift direction and kernel position. Additionally, a shift operation is required for each state. This gives a computational complexity proportional to $(k^2|A||D| + 1)|S|$, where D can be considered constant when the number of spatial dimensions and maximum movement range are below a fixed maximum. As a comparison, prior methods require one multiplication and one addition for each combination of state, action and kernel position, resulting in a total number of operations proportional to $k^2 \times |A| \times |S|$. This comparison shows that LCI-Net and prior methods have identical computational time complexity to prior methods in terms of state and action space size.

In general, the complexity of $|D|$ is linear in the transition range r and the number of dimensions, $dim(S)$. However, scalability can be further improved by restricting D to a subset of the set of all possible shifts. Our experimental results (Section 5.2) on a diverse set of domains indicate that certain restrictions do not detrimentally affect policy performance or learning convergence speed.

4 Applying LCI-Net

In this work, we use LCI-Net to replace the forward propagation module of QMDP-Net.

QMDP-Net's overall architecture consists of two components: Planning and Belief Update. The planning component approximates eq. (1) using QMDP heuristic. The Belief Update applies E2E-HF[7] to compute eq. (2). Both value iteration and belief update contains forward propagation operations. LCI-Net replaces the forward propagation module in both of those components of QMDP-Net.

4.1 Application to the Planning Component

QMDP approximates eq. (1) by assuming that the agent's state becomes fully observable after the first step. More precisely, it computes $V^*(b) = \max_{a \in A} \sum_{s \in S} b(s)[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')]$, where $V^*(s)$ is approximated via value iteration, i.e., via iterative computation of

$$V_{t+1}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V_t(s') \right] \tag{3}$$

until $t > T$ for a constant time limit T.

QMDP-Net adopted the value iteration implementation of VIN[21], where value iteration is implemented as a recurrent neural network. This network consists of a repeating block structure, in which each block represents a single step of value iteration and blocks can be stacked to arbitrary depth to produce any desired planning horizon. Each value iteration block contains a forward propagation module that computes the intermediate value $V'_{t+1}(s,a) = \sum_{s' \in S} T(s,a,s')V_t(s')$ of eq. (3). We replace this forward propagation module in each value iteration block with.

Each value iteration block takes as input a value image $V_t(s|\theta)$, and produces as output updated values based on one additional planning step, $V_{t+1}(s|\theta)$, with the input to the first block, $V_0(s|\theta)$, taken from the prediction of the immediate reward associated with each $s \in S$ provided by $f_R(.|\theta)$. In this application, LCI-Net takes $V_t(s|\theta)$ and produces $V'_{t+1}(s,a)$. The reward image $R(s,a)$ is then summed with $V'_{t+1}(s,a)$ to incorporate the immediate reward received at time step $t+1$, yielding $Q_{t+1}(s,a)$. $V_{t+1}(s|\theta)$ is then produced by selecting the action channel of Q_{t+1} with the greatest expected return. Fig. 2 illustrates this block.

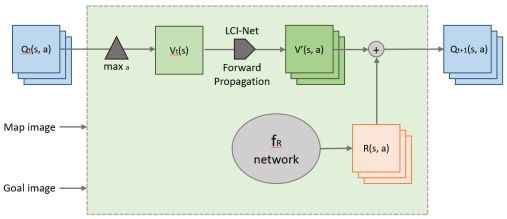

Fig. 2. Value Iteration network block. LCI-Net becomes its forward propagation module.

4.2 Application to the Belief Update Component

A POMDP agent maintains a belief, which is updated at each time step, as governed by eq. (2). In QMDP-Net, belief update uses a neural-network based histogram filter, called E2E-HF. The forward propagation module is used in this filter to compute in the intermediate value $b'_{t+1}(s,a) = \sum_{s \in S} T(s,a,s')b(s)$ in eq. (2). We replace this module in the belief update block with LCI-Net.

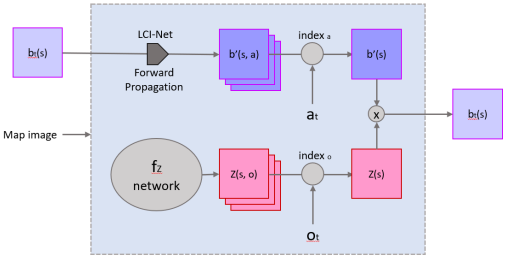

Fig. 3. Belief Update network block. LCI-Net becomes its forward propagation module.

This belief update block takes a prior belief b_t, an action a_t and an observation o_t as input, and produces the updated belief b_{t+1} as output, which is stored as the prior belief for the next action selection. In this application, LCI-Net is applied to the prior belief image to produce $b'_{t+1}(s,a)$, the belief propagated forward by one time step after action $a \in A$ is performed. This tensor is indexed based on the performed action a_t, with the channel corresponding to a_t retained, and all other channels discarded, giving $b'(s)$, the updated prior belief.

In parallel, the observation model $Z(s|o)$ produced by the model component network $f_Z(.|\theta)$ is indexed based on the perceived observation o_t, with the channel corre-

sponding to o_t selected to give $Z(s)$, the likelihood of each state based on the perceived observation. $b(s)$ and $Z(s)$ are then multiplied and normalised to give the posterior belief $b_{t+1}(s)$. Fig. 3 illustrates this Belief Update component.

5 Experiments

5.1 Experimental Setup

To evaluate the potential of LCI-Net in increasing the performance of combined learning and planning, we replace the forward propagation module of state-of-the-art QMDP-Net with LCI-Net, and compared this modified QMDP-Net (which we denote as LCI-Net for short) with the original QMDP-Net on a variety of partially observable stochastic environments. Results of LCI-Net are based on an implementation developed on top of the software released by the QMDP-Net authors, while QMDP-Net results are based on their released code.

Both networks are trained via imitation learning using the same set of expert trajectories, with the expert trajectories generated by applying the Q_{MDP} algorithm to manually constructed ground-truth POMDP models. Only trajectories where the expert was successful were included in the training set. The networks interact only with the expert trajectories and not with the ground-truth model. All hyper-parameters for both networks are set to match those used in the QMDP-Net experiments[9].

Training was conducted using GPU on an Nvidia GeForce RTX2070 GPU with 8GB of dedicated memory. The GPU is installed in a machine equipped with an Intel Xeon Silver 4110 CPU (8 cores, 16 threads at 2.10GHz) and 128GB of system RAM. We tested the networks on four domain types:

2D Dynamic Maze A navigation problem in a maze environment with structure that mutates during run-time in a way which qualitatively affects the optimum policy, designed to measure the robustness of a policy to dynamic environments. The robot must localise itself and navigate to the goal, while accounting for the possibility of environmental changes.

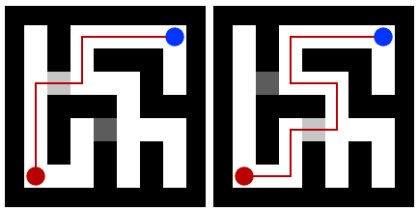

Fig. 4. Example of a 9×9 dynamic maze environment in both possible gate states. Light grey represents an open gate, dark grey a closed gate. The agent must navigate from the red circle to the blue circle. The red line denotes the optimal trajectory.

The robot is given a map of obstacle positions, a specified goal location, and initial belief distribution. No other environment information is given, and the POMDP model is not known a priori. At each time step, the robot selects a direction to move in. The outcomes of actions are probabilistic. Observations are received based on whether an obstacle is present in the adjacent cell in each of the "north", "south", "east" and "west" directions, with an independent fixed chance to receive an incorrect sensor reading for each direction.

To generate the dynamic maze layout, a maze is initially constructed using randomized Prim's algorithm. The maze is divided into 2 partitions, with 2 cells from the border selected to be gates. At each time step, exactly one gate is open and the gates will swap

from open to closed and vice versa with certain probability. During run time, the environment map provided to the agent is updated when a gate swap occurs. The start and goal position are selected such that a gate swap will cause the optimum solution to be qualitatively changed. Fig. 4 illustrates an example. Two variations of this scenario are evaluated:
V1: The network is trained using only expert trajectories from the static maze navigation task. The environment image provided in θ shows only the positions of current free spaces and current obstacles, without special marking for open or closed gates.
V2: The network is trained using trajectories based on an expert which plans on a dynamic ground truth POMDP model, allowing the expert to decide whether to wait for a nearby closed gate to open. The environment image received by the agent denotes the position of the gate which is currently closed. This may allow the agent to learn to intelligently decide whether to move or wait for the currently open gate to change. The open gate is not represented in the image. The networks are trained on a set of 9×9 dynamic maze environments containing 2000 environments with 5 trajectories per environment, and evaluated on both 9×9 and 29×29 dynamic mazes. Evaluation results are based on 1250 trials composed of 50 environments, 5 trajectories per environment and 5 repetitions per trajectory.

2D Navigation with Large Scale Realistic Environments A robot navigation problem in a general 2D grid setting with noisy state transitions and limited observations. The robot receives a map of obstacle positions, a specified goal location, and initial belief distribution. The POMDP model is not known a priori. At each time step, the robot selects a direction to move in, and receives a noisy observation indicating whether an obstacle is present in each direction. The networks are trained on artificial environments, with obstacle positions sampled at uniform random. Two different sizes of training environment are employed — 10×10 and 20×20. The 10×10 set contains 2000 environments with 5 trajectories per environment, while the 20×20 set contains 6000 environments with 5 trajectories per environment.

After training on the artificial environment set, evaluation is performed on environments modelled on the LIDAR maps from the Robotics Data Set Repository [6]. The robot receives an environment floor plan, a specified goal position and a randomly initialised belief distribution. No other information or model is provided. Three different maps are evaluated in both deterministic and stochastic form, each with dimensions on the order of 100×100. For the deterministic case, evaluation results are based on 100 trials for each map. For the probabilistic case, results are based on 150 trials per map.

3D Navigation of Multi-rotor Drone A navigation problem in 3 dimensional space with noisy state transitions and limited, unreliable observations, representing the task of control of autonomous multi-rotor drones through spaces with dense obstacles with limited sensing. The drone is given a 3D model of obstacle positions, a specified goal location, and initial belief distribution, and does not know the POMDP model a priori. The robot must localise itself and navigate to the goal.

The networks are trained on a set of artificial $7 \times 7 \times 7$ 3D environments comprising of 6000 environments with 5 trajectories per environment, with evaluation performed on both $7 \times 7 \times 7$ and $14 \times 14 \times 14$. Evaluation results are based on 1250 trials composed of 50 environments, 5 trajectories and 5 repetitions.

2D and 3D Grasping A robot gripper picks up randomly generated obstacles placed on a surface using a two-finger hand with observations received only via touch sensors mounted on the hand's fingertips. The agent receives a 2D image or 3D model of the shape object to be grasped and an additional image or model indicating which parts of the object can feasibly be grasped. The agent does not know its initial pose, and outcomes of transitions and sensor readings are probabilistic. We evaluate the networks on simplified variants of this task in both 2 and 3 dimensions. Figure 5 shows an example 3D grasping scenario.

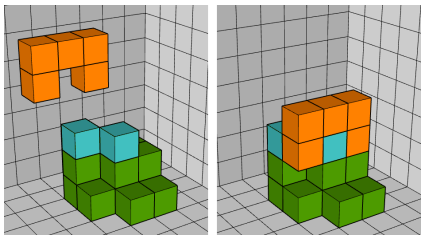

Fig. 5. Example of a 3D grasping task. Orange indicates the position of the grasping hand, and green and blue indicate an object to be grasped, where the blue areas are feasible grasp points. The left image shows a possible initial pose for the grasping hand, while the right image shows the hand grasping at feasible grasp point on the object.

The networks are trained on a fixed set of randomly generated objects placed in random positions on a table surface, and evaluated on a new set of previously unseen objects. In the 2D case, the training set comprises 80 objects, with 125 combinations of object position and initial gripper pose. In the 3D case, the training set is composed of 6000 object shapes, with 5 distinct positions for the obstacle and the gripper starting pose. Evaluation results are based on 1250 trials composed of 50 environments, 5 trajectories and 5 repetitions.

5.2 Results and Discussion

Our results demonstrate that LCI-Net, despite representing only a relatively small change in the network architecture of QMDP-Net, is able to deliver significant increases in performance and efficiency.

Table 1. Performance of LCI-Net and QMDP-Net on 2D dynamic maze. LCI-Net represents our network incorporating LCI-Net with all possible shift directions included, D indicates deterministic transitions and observations, while S represents stochastic transitions and observations. Traj Len is the average length of all successful trajectories, and is only directly comparable between cases where Success Rate is similar. Col Rate is the fraction of actions which resulted in collision over all trajectories, regardless of whether successful.

Environment	Network Type	Success Rate	Traj Len (95%CI)	Col Rate (95%CI)
Dynmaze V1 S 9x9	QMDP-Net	0.887	23.7 (± 1.1)	0.226 (± 0.009)
(trained on 9x9 S)	LCI-Net	0.942	20.2 (± 0.8)	0.195 (± 0.008)
Dynmaze V1 S 29x29	QMDP-Net	0.031	23.7 (± 1.1)	0.226 (± 0.009)
(trained on 9x9 S)	LCI-Net	0.302	20.2 (± 0.8)	0.195 (± 0.008)
Dynmaze V2 S 9x9	QMDP-Net	0.808	24.1 (± 1.2)	0.213 (± 0.008)
(trained on 9x9 S)	LCI-Net	0.978	18.7 (± 0.8)	0.123 (± 0.005)
Dynmaze V2 S 29x29	QMDP-Net	0.091	71.8 (± 0.7)	0.412 (± 0.009)
(trained on 9x9 S)	LCI-Net	0.590	59.7 (± 2.4)	0.228 (± 0.011)

Table 1 presents comparisons on the success rate, average number of steps, and collision rate of executing the policies generated by LCI-Net and by QMDP-Net for the dynamic maze environment tasks. The supplementary video shows a comparison of these policies. Table 2 shows the performance of LCI-Net on the same task where D is restricted to Non-Diagonal shift directions.

Table 2. Performance of LCI-Net with D restricted to Non-Diagonal directions on 2D dynamic maze tasks.

Environment	Success Rate	Traj Length (95%CI)	Collision Rate (95%CI)
Dynmaze V1 S 9x9	0.985	18.6 (± 0.7)	0.157 (± 0.007)
Dynmaze V1 S 29x29	0.743	18.6 (± 0.7)	0.157 (± 0.007)
Dynmaze V2 S 9x9	0.971	18.6 (± 0.8)	0.135 (± 0.006)
Dynmaze V2 S 29x29	0.604	60.7 (± 2.3)	0.148 (± 0.005)

A number of key conclusions can be drawn from these results. First, our network incorporating LCI-Net is able to produce consistently higher success rates and lower collision rates than QMDP-Net. In the cases where success rates are closest between the networks, LCI-Net produces average trajectory lengths which are near or below those produced by QMDP-net. This applies both when D includes all possible shifts and when D is selected to contain Non-Diagonal shifts only, with the the Non-Diagonal variant delivering comparable or better performance than the version which uses all possible shift directions.

Secondly, the size of the disparity between the performance of the architectures becomes dramatic when the learned models are generalised to larger environments. In Dynamic Maze V1, the success rate of QMDP-Net drops by more than 95 percent when environment size is increased to 29×29, while the ND variant of LCI-Net has its success rate reduced by less than 25 percent. The V2 maze variant gives similar results - the QMDP-Net success rate drops by almost 90 percent, while the ND version of LCI-Net drops by less than 40 percent.

This indicates that the introduction of LCI-Net greatly improves the generalisation capability of QMDP-Net, allowing effective policies to be found in large environments while requiring only expert trajectories on small environments for training. This result is likely enabled by the greater domain knowledge able to be encoded via the structure of the LCI-Net forward propagation. This represents a significant step towards making end-to-end learning for planning practical for real world applications.

Table 3. Performance of LCI-Net and QMDP-Net on 2D navigation benchmarks. Training is performed on a set of small artificially generated environments, while evaluation is performed on large environments based on LIDAR scans of buildings.

Environment	Network Type	Success Rate	Traj Len (95%CI)	Col Rate (95%CI)
Building 79 D	QMDP-Net	0.120	60.2 (± 18.5)	0.191 (± 0.067)
(trained on 10x10 D)	LCI-Net	0.870	73.7 (± 8.0)	0.031 (± 0.026)
Building 79 S	QMDP-Net	0.113	138.8 (± 40.2)	0.349 (± 0.044)
(trained on 10x10 S)	LCI-Net	0.567	139.0 (± 12.0)	0.068 (± 0.016)
Building 79 S	QMDP-Net	0.508	126.7 (± 12.3)	0.177 (± 0.027)
(trained on 20x20 S)	LCI-Net	0.664	126.9 (± 10.2)	0.050 (± 0.007)
Intel Labs D	QMDP-Net	0.120	102.8 (± 48.8)	0.059 (± 0.045)
(trained on 10x10 D)	LCI-Net	0.940	87.4 (± 10.0)	0.016 (± 0.018)
Intel Labs S	QMDP-Net	0.073	81.5 (± 47.5)	0.368 (± 0.043)
(trained on 10x10 S)	LCI-Net	0.547	138.8 (± 15.0)	0.067 (± 0.010)
Intel Labs S	QMDP-Net	0.468	138.6 (± 12.1)	0.177 (± 0.027)
(trained on 20x20 S)	LCI-Net	0.664	138.4 (± 10.0)	0.065 (± 0.012)
Hospital D	QMDP-Net	0.050	59.0 (± 63.4)	0.345 (± 0.069)
(trained on 10x10 D)	LCI-Net	0.500	74.6 (± 10.0)	0.060 (± 0.039)
Hospital S	QMDP-Net	0.093	104.5 (± 34.7)	0.337 (± 0.041)
(trained on 10x10 S)	LCI-Net	0.433	113.3 (± 14.8)	0.128 (± 0.025)
Hospital S	QMDP-Net	0.440	123.3 (± 12.2)	0.113 (± 0.020)
(trained on 20x20 S)	LCI-Net	0.552	107.1 (± 8.8)	0.083 (± 0.008)

Table 3 shows a comparison of the performance of policies produced by LCI-Net and by QMDP-Net for the 2D navigation tasks. The trends from the dynamic maze tasks continue. LCI-Net consistently produces a higher overall level of policy performance, with the distinction between the networks most pronounced when training is performed on the smaller 10×10 environments. The policy produced by QMDP-Net from the 10×10 returned a success rate of below 15% in each of the trialed environments, while the policy produced by LCI-Net achieves success rates above 50% for all but one of the environments, and above 80% for two environments.

Table 4. Performance of LCI-Net and QMDP-Net on 3D navigation of multi-rotor drone.

Environment	Network Type	Success Rate	Traj Len (95%CI)	Col Rate (95%CI)
Grid 3D 7x7x7 S	QMDP-Net	0.949	17.2 (± 0.9)	0.069 (± 0.005)
(trained on 7x7x7 S)	LCI-Net (ND)	0.950	16.1 (± 0.9)	0.066 (± 0.005)
Grid 3D 14x14x14 S	QMDP-Net	0.514	26.3 (± 0.8)	0.086 (± 0.006)
(trained on 7x7x7 S)	LCI-Net (ND)	0.729	26.8 (± 0.7)	0.063 (± 0.004)

Table 4 shows results for the 3D multi-rotor drone navigation task. Both network architectures are able to produce high rates of success and low rates of collision when evaluated on environments of the same size as used in training. Increasing the scale and complexity of the evaluation environment again shows an advantage in performance produced by LCI-Net.

Table 5. Performance of LCI-Net and QMDP-Net on grasping tasks.

Environment	Network Type	Success Rate	Traj Len (95%CI)	Col Rate (95%CI)
Grapser 2D 14x14	QMDP-Net	0.606	18.2 (± 1.2)	0.118 (± 0.008)
(trained on 14x14)	LCI-Net (ND)	0.700	19.0 (± 1.1)	0.116 (± 0.008)
Grapser 3D 7x7x7	QMDP-Net	0.883	10.2 (± 0.8)	0.163 (± 0.012)
(trained on 7x7x7)	LCI-Net (ND)	0.922	10.7 (± 0.8)	0.177 (± 0.013)
Grapser 3D 14x14x14	QMDP-Net	0.298	27.7 (± 2.0)	0.359 (± 0.014)
(trained on 7x7x7)	LCI-Net (ND)	0.319	29.9 (± 2.0)	0.409 (± 0.016)

Table 5 presents results for the object grasping tasks. Here, LCI-Net continues to produce higher success rates. Both networks are able to produce effective policies for 3D grasping on environments with the same dimensions as the training set, though generalising to larger environments is challenging for both architectures, with LCI-Net giving a small advantage in success rate.

Table 6 provides a comparison of the time and memory required for training for each environment between LCI-Net and QMDP-Net. The more expressive transition operator incorporated in LCI-Net introduces only a small amount of extra complexity. The additional time required per epoch of training is small in most cases, and is often compensated for by a decrease in the number of epochs of training required to reach convergence.

When only Non-Diagonal shifts are included in D, the complexity results are particularly promising. In some cases, LCI-Net with Non-diagonal shifts requires less time per epoch of training than QMDP-Net while converging in fewer epochs, resulting in a significant reduction in the total amount of time for training, while still producing policies which perform at a higher level than QMDP-Net policies.

In most cases, the additional amount of memory consumed by LCI-Net is negligible relative to total memory consumption. While scaling to larger environments results in

Table 6. Comparison of time and resources required for training in each environment between our network with LCI-Net and QMDP-Net. LCI-Net represents our network incorporating LCI-Net with all possible shift directions included, LCI-Net (ND) represents our network with only Non-Diagonal shift directions included. Time per epoch is in mm:ss format, while total train time is in hh:mm:ss format. Memory usage refers to the amount of GPU memory consumed - this is the only memory used for training.

Environment	Network Type	# Shift Directions	Time per epoch	Epochs to converge	Total Train Time	Memory Usage (MiB)
Grid 2D 20x20	QMDP-Net		3:08	612	31:57:36	2333
	LCI-Net	9	3:46	678	42:33:48	2609
	LCI-Net (ND)	5	2:44	624	28:25:36	2333
Dynmaze V1 9x9	QMDP-Net		0:20	997	5:32:20	785
	LCI-Net	9	0:28	996	7:44:48	789
	LCI-Net (ND)	5	0:21	828	4:49:48	789
Dynmaze V2 9x9	QMDP-Net		0:26	920	6:38:40	1553
	LCI-Net	9	0:31	613	5:16:43	1557
	LCI-Net (ND)	5	0:26	463	3:20:38	1557
Grid 3D 7x7x7	QMDP-Net		2:48	451	21:02:48	4893
	LCI-Net (ND)	7	2:26	609	24:41:54	4913
Grasper 2D 14x14	QMDP-Net		5:43	544	51:49:52	527
	LCI-Net (ND)	5	5:57	329	32:37:33	529
Grasper 3D 7x7x7	QMDP-Net		3:33	371	21:57:03	5405
	LCI-Net (ND)	7	3:10	304	16:02:40	5421

an increase in required memory, the rate of growth in memory consumption is very close to that of QMDP-Net.

6 Summary

Many neural network architectures for solving stochastic planning with partially unknown models based on end-to-end learning have been proposed. Across these architectures, there are a number of seemingly small components that have been used repeatedly, creating a great potential benefit in improving the efficiency of these components. Improvements in these components will likely improve the overall performance, similar to how improvement in "primitive" computations in motion planning improves the performance of the overall planning capability. Taking a step in this direction, this paper presents LCI-Net, a neural-network module that computes one-step information propagation governed by a learned stochastic model of the system's dynamics. It is a simple neural-network module that can be plugged into various neural network architectures. Evaluating LCI-Net on QMDP-Net[9], and hence on E2E-HF[7], on 2D and 3D navigation and grasping benchmarks indicate that LCI-Net creates significant gains in performance, with generalisation capability increased by a factor of up to 10.

7 Acknowledgements

Nicholas Collins is supported by an Australian Government Research Training Program (RTP) scholarship provided by the University of Queensland.

References

1. Neural networks, types, and functional programming. http://colah.github.io/posts/2015-09-NN-Types-FP/, published: 2019-09-03

2. François-Lavet, V., Bengio, Y., Precup, D., Pineau, J.: Combined reinforcement learning via abstract representations. In: AAAI. vol. 33, pp. 3582–3589 (2019)
3. Gupta, S., Davidson, J., Levine, S., Sukthankar, R., Malik, J.: Cognitive mapping and planning for visual navigation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jul 2017), http://dx.doi.org/10.1109/CVPR.2017.769
4. Haarnoja, T., Ajay, A., Levine, S., Abbeel, P.: Backprop KF: Learning discriminative deterministic state estimators. In: NIPS Conference (2016)
5. Hausknecht, M., Stone, P.: Deep recurrent Q-learning for partially observable MDPs. In: AAAI 2015 Fall Symposium (2015)
6. Howard, A., Roy, N.: The robotics data set repository (radish) (2003), http://radish.sourceforge.net/
7. Jonkowski, R., Brock, O.: End-to-end learnable histogram filters (2017)
8. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial intelligence 101(1-2), 99–134 (1998)
9. Karkus, P., Hsu, D., Lee, W.S.: QMDP-net: Deep learning for planning under partial observability. In: NIPS Conference (2017)
10. Karkus, P., Hsu, D., Lee, W.S.: Particle filter networks with application to visual localization. In: CoRL Conference (2018)
11. Karkus, P., Ma, X., Hsu, D., Kaelbling, L.P., Lee, W.S., Lozano-Perez, T.: Differentiable algorithm networks for composable robot learning. In: Robotics: Science and Systems (2019)
12. Lisa Lee and Emilio Parisotto and Devendra Singh Chaplot and Eric Xing and Ruslan Salakhutdinov: Gated Path Planning Networks. In: ICML Conference (2018)
13. Littman, M.L., Cassandra, A.R., Kaelbling, L.P.: Learning policies for partially observable environments: Scaling up. In: ICML (1995)
14. Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., Hadsell, R.: Learning to navigate in complex environments. In: ICLR Conference (2016)
15. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518, 529 EP (Feb 2015), https://doi.org/10.1038/nature14236
16. Oh, J., Guo, X., Lee, H., Lewis, R.L., Singh, S.: Action-conditional video prediction using deep networks in atari games. In: NIPS Conference. pp. 2863–2871 (2015)
17. Oh, J., Singh, S., Lee, H.: Value prediction network. In: NIPS Conference. pp. 6118–6128 (2017)
18. Okada, M., Rigazio, L., Aoshima, T.: Path integral networks: End-to-end differentiable optimal control (2017)
19. Shankar, T., Dwivedy, S.K., Guha, P.: Reinforcement learning via recurrent convolutional neural networks. In: ICPR Conference. pp. 2592–2597 (Dec 2016)
20. Sondik, E.: The optimal control of partially observable Markov processes. Ph.D. thesis, Stanford University (1971)
21. Tamar, A., Wu, Y., Thomas, G., Levine, S., Abbeel, P.: Value iteration networks. IJCAI Conference (Aug 2017)
22. Wahlström, N., Schön, T.B., Deisenroth, M.P.: Learning deep dynamical models from image pixels. In: The 17th IFAC Symposium on System Identification (SYSID) (2015)