Partially Observable Planning and Learning for Systems with Non-Uniform Dynamics

Nicholas Collins
The University of Queensland
n.collins@uq.edu.au

Hanna Kurniawati
Australian National University
hanna.kurniawati@anu.edu.au

Abstract

We propose a neural network architecture, called *TransNet*, that combines planning and model learning for solving Partially Observable Markov Decision Processes (POMDPs) with non-uniform system dynamics. The past decade has seen a substantial advancement in solving POMDP problems. However, constructing a suitable POMDP model remains difficult. Recently, neural network architectures have been proposed to alleviate the difficulty in acquiring such models. Although the results are promising, existing architectures restrict the type of system dynamics that can be learned - that is, dynamics must be the same in all parts of the state space. TransNet relaxes such a restriction. Key to this relaxation is a novel neural network module that classifies the state space into classes and then learns system dynamics of the different classes. TransNet uses this module together with the overall architecture of QMDP-Net[Karkus *et al.*, 2017] to allow solving POMDPs that have more complex dynamic models while maintaining efficient data requirement. Evaluation on typical benchmarks in robot navigation with initially unknown system and environment models indicates that TransNet substantially outperforms the quality of the generated policies and learning efficiency of the state-of-the-art method QMDP-Net.

1 Introduction

Sequential decision making under uncertainty is both critical and challenging. Partially Observable Markov Decision Processes (POMDPs) are the general and systematic frameworks for computing such decision making problems. Although finding optimal strategies under the POMDP framework is computationally intractable, advances have been made in computing approximately optimal strategies[Kurniawati *et al.*, 2008; Silver and Veness, 2010; Somani *et al.*, 2013]. We now have algorithms that can find near optimal strategies within reasonable time and have been applied to solve various realistic robotics problems (e.g., [Bouton *et al.*, 2017; Chen *et al.*, 2018; Hoerger *et al.*, 2019]).

With POMDP solving becoming practical and POMDPs becoming used in practice, the problem of generating a good POMDP model for a given problem becomes increasingly important. A POMDP model is defined by six components: The states the system can be in, the available actions, the observations it can perceive, system dynamics representing uncertainty in the effect of actions, an observation function which represents sensing uncertainty, and a reward function from which the objective function is derived. While the first three components are easy to define, the last three are more difficult due to uncertainty in the system and imperfect or even non-existent measurements to assess them.

Many machine learning techniques have been proposed to alleviate this difficulty[Ghavamzadeh *et al.*, 2015; Arulkumaran *et al.*, 2017]. They can be divided into two broad classes. First is model-free, where the system learns a direct mapping from environment information to strategies, bypassing model generation. Second is model-based, where the system first learns the model, and strategies are generated by applying model-based planning techniques to this model.

Recently, deep neural networks have been proposed to combine model-free learning and model-based planning[Karkus *et al.*, 2017; Tamar *et al.*, 2017]. These works learn a direct mapping from environment information to strategies. However, internally these methods learn a POMDP model (or in the case of [Tamar *et al.*, 2017], an MDP model - a sub-class of POMDP where state is fully observable) and use a planning module, embedded inside the neural network, to generate the strategy. The objective of the model learning component here is not to generate the most accurate model, but rather

to generate a *useful* approximate model that will maximise policy performance when used together with the embedded planning algorithm. The results have been promising.

However, the above networks assume the transition function in the (PO)MDP problem to be independent of the states of the system. This assumption means the system dynamics is assumed to be the same everywhere, regardless of the geometry of the underlying environment, which often limits the expressiveness of the model and restricts the effectiveness of planning. To relax this assumption, we propose a novel neural network architecture, TransNet.

Key to TransNet is a differentiable neural network module that learns non-uniform transition dynamics efficiently by assuming that states with similar local characteristics have similar dynamics. This module divides the state space into classes, where each class corresponds to a unique transition function. The transition probabilities for each class are then represented by channels of a kernel in a convolution layer. This technique allows distinct transition dynamics to be applied to states with different local characteristics while still allowing the use of existing efficient convolutional network implementations. TransNet uses this novel neural network module together with the overall architecture of state-of-the-art QMDP-Net to solve POMDPs with a priori unknown model and non-uniform transition dynamics.

Simulations on various navigation benchmarks with and without dynamic elements indicate that compared to QMDP-Net, TransNet requires substantially less training time and data to produce policies with better quality: In some cases, TransNet uses less than 20% of the training data used by QMDP-Net to generate policies with similar quality. Our results also indicate that TransNet provides substantially better generalization capability than QMDP-Net.

2 Background

2.1 POMDP Framework

Formally, a POMDP[Kaelbling *et al.*, 1998] is described by a 7-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 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 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 sensing uncertainty. After each step, the agent receives reward $R(s, a)$, if it takes action a from state s.

Due to uncertainty in the effect of action and in sensing, the agent never knows its exact state. Instead, it maintains an estimate of its current state in the form of a *belief* b, which is a probability distribution over S. At the end of each step, the agent updates its belief in a Bayesian manner, based on the belief at the beginning of the step along with the action and observation that have been performed and perceived in this step.

The objective of a POMDP agent is to maximize its expected total reward (*value function*), by following the best policy at each time step. A *policy* is a mapping from beliefs to actions. Each policy π induces a value function $V_\pi(b)$ for any $b \in \mathbb{B}$, which is computed as:

$$V_\pi(b) = \sum_{s \in S} R(s, \pi(b))\ b(s) + \gamma \sum_{o \in O} P(o|b', \pi(b))\ V_\pi(b') \tag{1}$$

The notation b' represents the new belief of the agent after it performs action $\pi(b) \in A$ and perceives observation o afterwards. It is computed as $b'(s') = \eta \sum_{o \in O} \sum_{s \in S} Z(s', \pi(b), o)\ T(s, \pi(b), s')\ b(s)$ (η is a normalizing factor). When the planning horizon is infinite, to ensure the problem is well defined, rewards at subsequent time steps are discounted by a constant factor $\gamma \in (0, 1)$. The best policy π^* is one that maximizes the value function at each belief b.

2.2 Related Work

There is a growing body of works that apply model-free deep learning to solve large scale POMDPs when the model is not fully known. For instance, [Hausknecht and Stone, 2015] implemented a variation of DQN [Mnih *et al.*, 2015] which replaces the final fully connected layer with a recurrent LSTM layer to solve partially observable variants of Atari games. The work in [Mirowski *et al.*, 2016] 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.

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. [Tamar *et al.*, 2017] 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 [Okada *et al.*, 2017] 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 underlying state is fully observable.

By combining the ideas in the above work with recent work on embedding Bayesian filters in deep neural networks[Jonkowski and Brock, 2017; Haarnoja *et al.*, 2016; Karkus *et al.*, 2018], one can develop neural network architectures that combine model-free learning and model-based planning for POMDPs. For instance, [Shankar *et al.*, 2016] implemented a network which implements an approximate POMDP algorithm based on Q_{MDP} [Littman *et al.*, 1995] 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, [Karkus *et al.*, 2017] 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. However, the embedded model is restricted to using a simple transition model which assumes all states have the same transition dynamics. The transition function is represented as a kernel whose depth is the same as the size of the action space. The same learned kernel is applied to each state in the state space. This representation of the transition function enables the dynamics learned for one state to be generalised to other states, reducing the amount of training data needed to learn transition dynamics for all states. But as a result, QMDP-Net cannot represent non-uniform transition dynamics. TransNet relaxes this restriction, while maintaining data efficiency.

3 TransNet

TransNet learns 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. TransNet assumes that 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. TransNet learns the parameterized transition, observation, and reward functions suitable to generate a good policy for the set of scenarios in $\mathcal{W}_\Theta$, as it learns the policy.

Similar to QMDP-Net, TransNet's overall structure is a Recurrent Neural Network with two interleaving blocks: Planning and Belief update. Figure 1(a) illustrates this network. However unlike QMDP-Net, in each block, TransNet uses a neural network module as described in the following subsection to learn a transition function that depends on both actions and local characteristics of the states, rather than actions alone, thereby allowing more expressive POMDP models to be learnt, while maintaining data efficiency.

3.1 Learning Non-Uniform Transition Dynamics

Key to TransNet is a neural network module for learning the transition function of a set of parameterized POMDPs. Suppose $M(\theta) = (S, A, O, f_T(.|\theta), f_Z(.|\theta), f_R(.|\theta))$ is the POMDP problem that corresponds to a scenario $W(\theta) \in \mathcal{W}_\Theta$. To learn the transition function $f_T(.|\theta)$, the neural network module represents $f_T(.|\theta)$ by a combination of a learned kernel and a classification function. The classification function $c(s|\theta)$ is a surjection that maps each state $s \in S$ to a class index, based on features of the parameter θ. The kernel represents the probability of transitioning into each of the states in a local neighbourhood for each action $a \in A$ and each class, with separate channels representing different pairs of actions and classes. The pair of action and class index is then used to select the suitable kernel channel.

Two properties are desirable for the classification function. First, states with similar local characteristics should map to the same class, and states with highly dissimilar characteristics should map to different classes. Second, the number of distinct state classes produced by the classification should be large enough to represent the important distinct modes of the transition dynamics, but small enough to ensure that information learned about the dynamics of one state is allowed to generalise to as many other appropriate states as possible.

To generate the above desirable properties, in this work, the classification function is constructed by selecting a number of features of the scenario parameter θ which correspond to the local features. The classification function $c(s|\theta)$ then maps each state $s \in S$ to a class index based on the combination of feature values of the state s. Let N be the number of features and $f_1(s)...f_N(s)$ be the values of the features of state $s \in S$. The classification function $c(s)$ is:

$$c(s) = \sum_{1 \leq i \leq N} (M+1)^{i-1} f_i(s)$$

where M is the maximum value of any feature of any state $s \in S$. The class index $c(s)$ of state s indicates the transition model to use at s. We denote the image of this function, which represents the set of possible state classes, as C.

As an example, in a 2D robot navigation problem where θ includes an image indicating whether each cell in

the environment is an obstacle (represented by 1) or free space (represented by 0), the features can be selected to be the values of the cells to the *north*, *south*, *east* and *west* of the current cell based on this image. The function $c(s)$ is then defined as $f_{\text{North}}(s) + 2f_{\text{South}}(s) + 4f_{\text{East}}(s) + 8f_{\text{West}}(s)$. When a state s is blocked by obstacles in only its north and east side for instance, $c(s) = 1 + 0 + 4 + 0 = 5$. Of course, the image does not have to be binary. It may also represent information such as terrain types, obstacle types with different elasticity, areas of the environment which are subject to change over time, etc., allowing this representation to generalise to a wide range of scenarios.

To avoid creating a bottleneck in the network, the classification function is implemented as a matrix operation in existing tensor libraries, allowing an image representing the state classification of every state in the state space to be computed efficiently for all states at once. Furthermore, a one-hot mapping is applied to the output of this function, which is then used to index into the channel corresponding to the local characteristics of each state using efficient matrix multiplication and summation operations. An illustration of TransNet for a problem where the state space S consists of two state variables, whose size is n and m, respectively, is shown in Figure 1(b).

The above manual selection of features and algorithmic classification could be replaced by an additional convolutional neural network, allowing important features which influence transition dynamics to be learned adaptively.

Note that the effect of the TransNet architecture is not equivalent to simply applying QMDP-net to a state space augmented with an additional dimension representing class. While augmenting the state space with a class variable would permit different transition models to be learned for different classes, this approach would not allow classes to be assigned to states at run time based on observations about the given map, but would rather only learn an approximate assignment of classes based on the class distribution of the training set.

Note also that this module is general enough that it can be combined with any neural network architecture that embed POMDP/MDP planning with initially unknown transition function. However, TransNet combines this module with QMDP-Net and embeds the module within every planning and belief update block. The following two subsections provide more details on this embedding.

3.2 Planning

The planning component of TransNet 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 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.

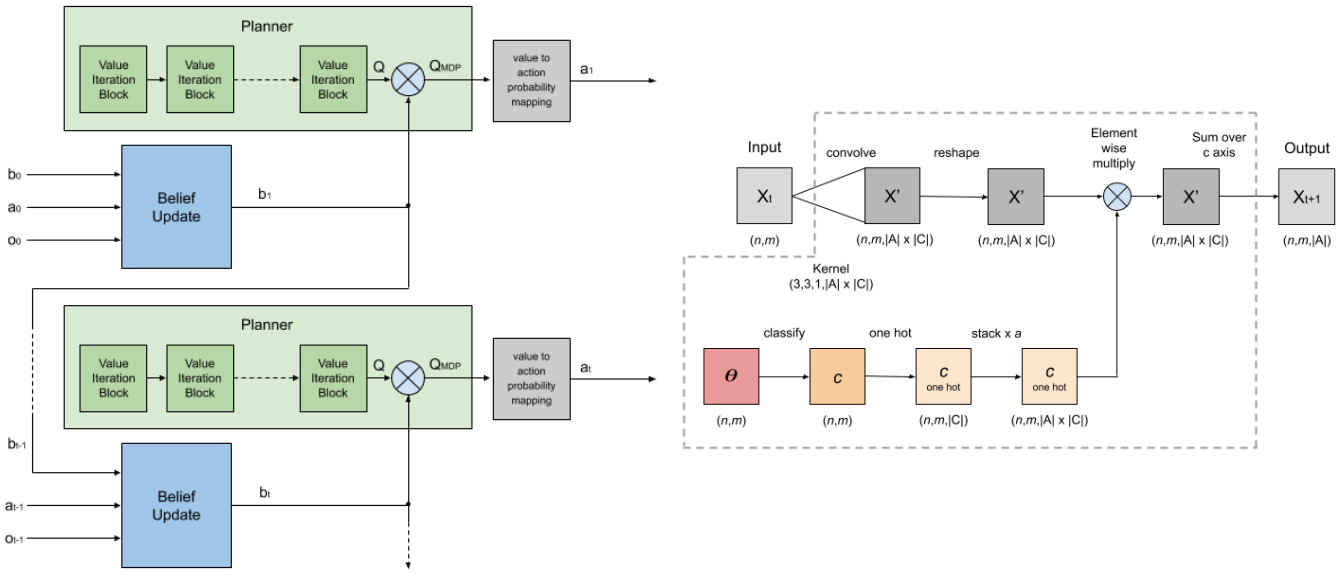
TransNet convolves the input with the neural network module for learning transition function. This module has one output channel for each pair (a, c), where $a \in A$ and $c \in C$. The result of the convolution is a layer that represents the Q-values for each combination of state, action and class index. Since for any scenario with parameter $\theta \in \Theta$, $c(s|\theta)$ is a surjection, we only need to select Q-values for the class that matches with $c(s|\theta)$. Therefore, the Q-values are multiplied with the one-hot representation of the state class image, before being summed over the axis corresponding to c. This has the effect of selecting the correct Q-values for the current θ, and discarding all other invalid Q-values. These corrected Q-values are re-weighted by the belief. The maximum of these corrected Q-values over all $a \in A$ is then selected via a max-pooling layer to produce the updated value $V_{t+1}(s|\theta)$. The architecture of this block is illustrated in Figure 2(a).

This implementation is a compromise, which sacrifices space complexity efficiency by computing and temporarily storing Q-values for classes which do not match $c(s|\theta)$ in order to facilitate the use of existing highly optimised implementations of convolutional network layers, without which training the network is infeasible.

3.3 Belief Update

A POMDP agent maintains a belief, which is updated at each time step using a Bayesian filter. To this end, TransNet interleaves the planning block with the belief update block. The belief update block takes a prior belief b_t, action a_t and 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.

To compute b_{t+1}, TransNet convolves b_t with the neural network module for learning transition function. The resulting convolution is an image with one channel for each pair (a, c), where $a \in A$ and $c \in C$, representing the updated probability of being in each state $s \in S$ for each combination of action and class index. The one-hot representation of the classes is used to select only the values for which class matches $c(s)$. A one-hot representation of the action a_t applied at time t is then used to select the values for which action matches a_t. The resulting belief represents the belief after accounting for the effect of the transition dynamics, notated as b'. A one-hot representation of the received observation o_t is used to index into the observation model image predicted by



(a) Overall architecture.

(b) Key contribution of TransNet: A module that learns non-uniform transition function.

Figure 1: TransNet

f_Z to produce an image indicating the predicted probability of receiving o_t for each state $s \in S$. Finally this is used to weight b' to produce the complete updated belief image, b_{t+1}. The architecture of a belief update block is shown in Figure 2(b).

4 Experiments

4.1 Experimental Setup

To understand the practical performance of TransNet, we compared TransNet with state-of-the-art QMDP-Net. TransNet's results 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[Karkus *et al.*, 2017].

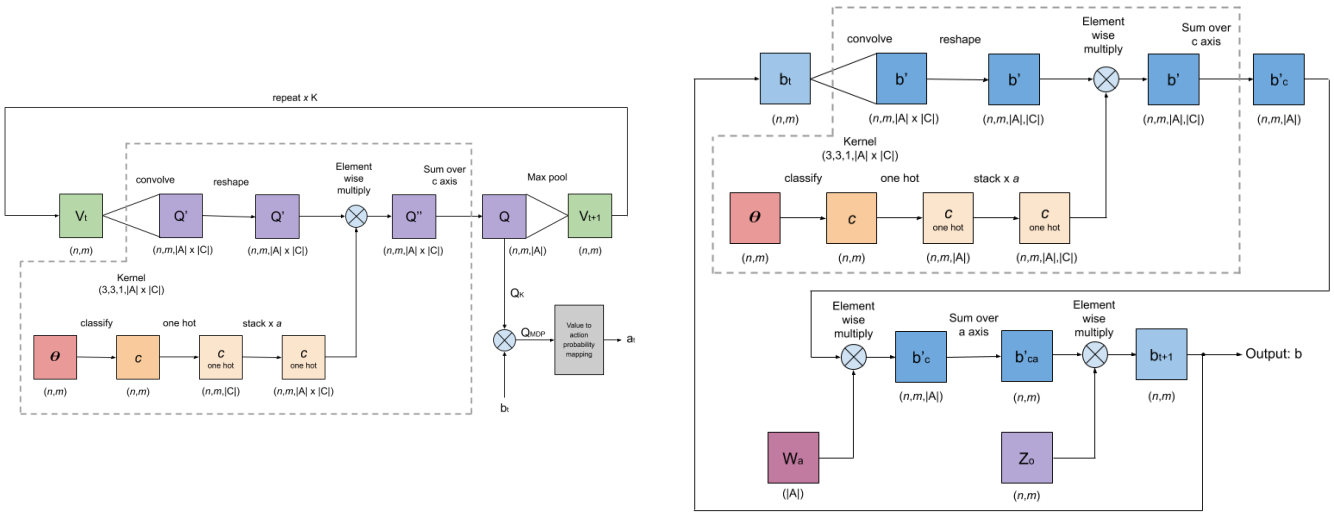
Training was conducted using CPU only on a machine with Intel Core-i7 7700 processor and 8GB RAM. We tested the networks on four domains:

Gridworld Navigation: A robot navigation problem in a general 2D grid setting with noisy state transitions and limited observations. The robot is given a map of obstacle positions, a specified goal location, and initial belief distribution. The robot must localise itself and navigate to the goal. 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 of the "north", "south", "east" and "west" directions. The obstacle configuration is generated uniformly at random, with the constraint imposed that all non-obstacle cells are mutually reachable via some path.

Maze Navigation: Similar to the gridworld navigation task, but with obstacle configuration generated using randomized Prim's algorithm. This results in expert trajectories typically being longer than in the general grid domain requiring longer term planning. This environment is also highly dependent on the planner's ability to identify dead-end passages.

Dynamic Maze Environments: 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.

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. The start and goal position are selected such that a gate swap will cause the optimum solution to be qualitatively changed. Figure 3 illustrates an example. Two variations of this scenario are evaluated:



(a) A planning block of TransNet

(b) A belief update block of TransNet

Figure 2: TransNet architecture. The part of TransNet that learns the transition function is marked by dashed-lines.

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 open. This may allow the agent to learn to intelligently decide whether to move or wait for the currently open gate to change. The closed gate is not represented in the image.

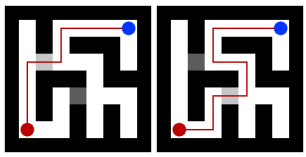


Figure 3: Example of a 9×9 dynamic maze environment in both possible gate states. Light grey represents an open gate, dark grey a closed gate. The agent must navigate from the red circle to the blue circle. The red line denotes the optimal trajectory.

Large Scale Realistic Environments: A navigation problem in realistic environments modelled on the LIDAR maps from the Robotics Data Set Repository [Howard and Roy, 2003] with noisy actions and limited, unreliable observations. The network is trained on a set of randomly generated 10×10 stochastic grid environments, with the resulting policy then applied to the realistic environments, which have dimensions in the order of 100×100.

4.2 Results and Discussion

Table 1 presents comparisons on the success rate, average number of steps, and collision rate of executing the policies generated by TransNet and QMDP-Net, as well as the policy produced by the expert agent used to generate the training trajectories, based on the Q_{MDP} algorithm applied to a perfect ground-truth model. Training is conducted until convergence, but policies are outputted at a regular interval of 50 epochs. Training uses 10,000 different scenarios, comprising of 2,000 different environments and 5 different trajectories per environment. Policy evaluation is conducted on 500 different scenarios, comprising of 100 new environments and 5 different trajectories per environment.

The results indicate that TransNet consistently produced substantially better policies than QMDP-Net and out-performs the training expert trajectories more consistently than QMDP-Net. The left side of Table 1 presents the results when training is run until convergence and comparison with the expert trajectory. In most cases, the number of epochs required to achieve convergence is lower in TransNet than in QMDP-Net. Moreover, compared to QMDP-Net, TransNet converges to policies with better quality. The right side of Table 1 presents the results where the training time are similar, giving slightly longer time to QMDP-Net. They indicate that although TransNet requires more training time per epoch than QMDP-Net, TransNet uses less time to generate policies with better quality.

The results also demonstrate TransNet is significantly more robust than QMDP-Net in dynamic environments. The success rate and collision rate of TransNet are not

Table 1: Performance comparison of TransNet and QMDP-Net. Expert is the QMDP algorithm. D indicates deterministic, S indicates stochastic. Epochs and time are the number of epochs and training time taken to generate the policy. SR is the success rate (in %) over all trials. TL is the average number of steps for successful trials. CR is the collision rate (in %) over all steps. 95% CI is the 95% confidence interval. Note that the number of steps required for completion is only directly comparable when success rates are similar.

		Converged Policy				Policy after Similar Training Time				
Domain	**Agent**	**Epochs**	**SR**	**TL (95% CI)**	**CR (95% CI)**	**Time (s)**	**Epochs**	**SR**	**TL**	**CR**
Grid 10x10 D	Expert		95.0	7.4 (±0.23)	0.0 (±0.0)					
	QMDP-net	248	100.0	7.5 (±0.20)	0.2 (±0.8)	1,420	100	81.7	13.0	7.1
	TransNet	328	100.0	7.5 (±0.19)	0.0 (±0.2)	1,310	50	89.8	8.6	7.5
Grid 10x10 S	Expert		98.0	15.5 (±1.45)	6.8 (±0.5)					
	QMDP-net	754	95.0	15.1 (±1.05)	13.9 (±1.8)	3,993	100	62.4	20.9	36.5
	TransNet	543	99.8	14.1 (±0.93)	10.0 (±1.2)	3,092	50	96.1	14.8	13.2
Maze 9x9 S	Expert		88.4	15.5 (±1.14)	10.5 (±0.7)					
	QMDP-net	1,086	73.6	23.8 (±1.90)	29.8 (±1.8)	2,940	100	69.1	20.8	31.2
	TransNet	837	97.8	15.6 (±0.94)	15.9 (±1.3)	2,257	50	83.0	18.7	23.6
Dynamic Maze V1 9x9 S	Expert		85.2	23.3 (±1.84)	13.1 (±0.9)					
	QMDP-net	1,565	71.0	25.8 (±2.21)	33.9 (±2.3)	2,982	100	62.1	24.7	32.8
	TransNet	1,171	97.6	18.6 (±1.02)	16.4 (±1.5)	2,289	50	67.7	23.7	30.7
Dynamic Maze V2 9x9 S	Expert		89.8	19.2 (±1.17)	11.8 (±0.9)					
	QMDP-net	934	66.8	22.1 (±1.85)	27.3 (±2.0)	8,129	250	53.7	24.9	30.4
	TransNet	1,122	87.6	19.1 (±1.45)	15.5 (±1.1)	7,902	50	63.5	22.6	20.8

substantially degraded by the introduction of dynamic environment elements, and performance remains at or above the level of the QMDP expert trajectories.

Another key result is that TransNet was able to consistently produce a higher success rate than the expert agent on 4 out of the 5 evaluated domains, with near equal performance on the remaining domain. As the expert uses a perfect model, it represents the best performance that can be achieved by the particular planner with the most accurate learned model possible. This demonstrates that TransNet is able to produce policies of a quality level which is not attainable through conventional model-based learning.

Table 2: Comparison of the converged policy of TransNet and QMDP-Net on Grid 10x10 S over different sizes of training set.

Trajectories	**Agent**	**SR**	**TL (95% CI)**	**CR (95% CI)**
2000	QMDP-Net	70.4	21.5 (±1.95)	32.0 (±2.2)
	TransNet	98.2	15.3 (±1.11)	11.2 (±1.3)
10000	QMDP-Net	95.0	15.1 (±1.05)	13.9 (±1.8)
	TransNet	99.8	14.1 (±0.93)	10.0 (±1.2)
50000	QMDP-Net	97.2	16.2 (±0.86)	7.9 (±1.0)
	TransNet	99.2	15.4 (±0.95)	6.8 (±0.7)

Table 2 presents a comparison of the performance of TransNet and QMDP-Net in a stochastic grid environment when trained on sets of expert trajectories of different sizes.

The results indicate TransNet significantly reduces data requirements. TransNet achieves a 98% success rate after training with 2,000 scenarios. In contrast, QMDP-Net requires 50,000 scenarios to attain a comparable rate of success in this domain. The reduced data requirements enable TransNet to be more practical for applications where acquiring training data is difficult or costly, such as when training data must be collected through interaction with a physical system.

Table 3: Comparison of the converged policy generated by TransNet and QMDP-Net trained on Grid 10x10 D for (deterministic cases) and Grid 10x10 S for (for stochastic cases) and evaluated on large scale realistic environments derived from LIDAR datasets.

Domain	**Agent**	**SR**	**TL**	**CR**
Intel Lab 101x99 D	QMDP-Net	40.0	100.0	6.6
	TransNet	96.0	94.3	1.2
Intel Lab 101x99 S	QMDP-Net	4.0	90.0	37.2
	TransNet	68.0	129.2	3.7
Building 079 145x57 D	QMDP-Net	56.0	70.8	22.5
	TransNet	78.0	65.2	4.8
Building 079 145x57 S	QMDP-Net	24.0	122.3	43.0
	TransNet	52.0	107.5	7.9
Hospital 193x104 D	QMDP-Net	14.0	85.1	28.6
	TransNet	84.0	91.2	3.9
Hospital 193x104 S	QMDP-Net	24.0	119.5	28.6
	TransNet	52.0	193.3	4.2

Table 3 presents the generalization capability of TransNet, compared to QMDP-Net. It compares the performance when networks trained on small artificially generated environments are evaluated on large scale realistic environments: Intel Lab corresponds to the Intel Research Lab dataset, Building 079 corresponds to the Freiburg Building 079 dataset, and Hospital corresponds

to the Freiburg University Hospital dataset. To evaluate scenarios, we ran 25 trials per environment. In the work of [Karkus *et al.*, 2017], QMDP-Net was demonstrated to produce high rates of success on deterministic large scale environments when trained on expert trajectories in 30×30 random grids. Here, we trained both TransNet and QMDP-Net on 10×10 random grids and evaluated in both deterministic and stochastic cases of realistic environments.

The results indicate TransNet substantially improves generalization capability. Local characteristics of states in the same class of problems (e.g., robot navigation in partially observed scenarios) tend to remain the same, even though the global complexity are totally different. Therefore, by learning separate transition functions based on local characteristics of the states, TransNet can generate policies that generalize well.

5 Conclusion

TransNet is a deep recurrent neural network for computing near optimal POMDP policies when the transition, observation, and reward functions are a priori unknown. The key novelty of TransNet is a relatively simple neural network module that can learn non-uniform transition function efficiently. Experiments on navigation benchmarks indicate that TransNet consistently out-performs state-of-the-art QMDP-Net. Moreover, results also indicate that TransNet can generalize better and substantially reduce the amount of training data and time required to reach certain performance.

This work suggests that a relatively simple neural network module can help embed more sophisticated models into deep neural networks, which then lead to substantial improvement for planning in stochastic domain. It is interesting to understand further how more sophisticated planning and learning components could help further scaling up of our capability in computing near optimal policies for decision making in stochastic domain.

References

[Arulkumaran *et al.*, 2017] Kai Arulkumaran, Marc Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[Bouton *et al.*, 2017] Maxime Bouton, Akansel Cosgun, and Mykel J Kochenderfer. Belief state planning for autonomously navigating urban intersections. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 825–830. IEEE, 2017.

[Chen *et al.*, 2018] M. Chen, S. Nikolaidis, H. Soh, D. Hsu, and S. Srinivasa. Planning with trust for human-robot collaboration. In *Proc. ACM/IEEE Int. Conf. on Human-Robot Interaction*, 2018.

[Ghavamzadeh *et al.*, 2015] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[Haarnoja *et al.*, 2016] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop KF: Learning discriminative deterministic state estimators, 2016.

[Hausknecht and Stone, 2015] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs, 2015.

[Hoerger *et al.*, 2019] Marcus Hoerger, Hanna Kurniawati, and Alberto Elfes. POMDP-based candy server: Lessons learned from a seven day demo. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS)*, 2019.

[Howard and Roy, 2003] Andrew Howard and Nicholas Roy. The robotics data set repository (radish), 2003.

[Jonkowski and Brock, 2017] Rico Jonkowski and Oliver Brock. End-to-end learnable histogram filters, 2017.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[Karkus *et al.*, 2017] Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-net: Deep learning for planning under partial observability, 2017.

[Karkus *et al.*, 2018] Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter networks with application to visual localization, 2018.

[Kurniawati *et al.*, 2008] 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*, 2008.

[Littman *et al.*, 1995] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, 1995.

[Mirowski *et al.*, 2016] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments, 2016.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin

Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529 EP, Feb 2015.

[Okada *et al.*, 2017] Masashi Okada, Luca Rigazio, and Takenobu Aoshima. Path integral networks: End-to-end differentiable optimal control, 2017.

[Shankar *et al.*, 2016] T. Shankar, S. K. Dwivedy, and P. Guha. Reinforcement learning via recurrent convolutional neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2592–2597, Dec 2016.

[Silver and Veness, 2010] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Proc. Neural Information Processing Systems*, 2010.

[Somani *et al.*, 2013] A. Somani, N. Ye, D. Hsu, and W.S. Lee. DESPOT: Online POMDP Planning with Regularization. In *Proc. Neural Information Processing Systems*. 2013.

[Tamar *et al.*, 2017] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Aug 2017.