Learning to Plan with Logical Automata

Brandon Araki
MIT
araki@mit.edu

Kiran Vodrahalli
Columbia University
knv@columbia.edu

Cristian-Ioan Vasile
MIT
cvasile@mit.edu

Daniela Rus
MIT
rus@csail.mit.edu

Abstract

We present a framework for combining imitation learning and logical automata and introduce the Logic-based Value Iteration Network (LVIN) model. By appending a 'logical' dimension to the state space of the environment, LVIN can recover and incorporate the transition matrix of a finite state automaton derived from a Linear Temporal Logic formula into the policy learned through imitation. This approach improves upon the original VIN in several ways: LVIN (1) is capable of learning logic corresponding to long sequences of steps, (2) adapts easily to new specifications, and (3) is amenable to correction after faulty expert demonstrations.

1 Introduction

In the imitation learning (IL) problem, learning agents receive access to experts who take good actions and can be queried in order to learn a policy for solving an unknown Markov Decision Process (MDP) (Abbeel and Ng, 2004; Daumé III et al., 2009; Ross et al., 2011). IL has been very successful in solving tasks as diverse as camera control, speech imitation, and self-driving cars (Yue and Le, 2018). However, the expert formalism requires new sets of expert demonstrations in order to perform new tasks. Additionally, these methods rely on the quality of the experts, and most IL methods require an assumption that the experts make no errors. Therefore, we ask

1. How can expert demonstrations for a single task generalize to much larger groups of tasks?
2. What if the experts are unreliable and err?

We provide an answer to these questions by applying methods from formal verification in a learning setting. We first learn interpretable high-level expert strategies and incorporate them in a learned policy. Then, we manipulate the expert strategies to produce new desired behaviors with the modified policy. As one application, we fix expert mistakes without re-training on any new data. Our approach also solves tasks requiring long sequences of accurate actions, where we demonstrate that standard learning approaches often fail.

In the robotics and control community, temporal logic languages such as Linear Temporal Logic (LTL) are used to unambiguously specify complex tasks, and a large class of these specifications can be directly translated into finite state automata (FSAs). Such FSAs can be thought of as high-level directives on accomplishing a task. Since an FSA is also an MDP, taking the product of the FSA and the MDP creates a product automaton (PA). When the MDP and LTL specification are known, one can find the optimal policy over the PA with standard planning methods (Vasile et al., 2016, 2017). In this paper, we extend this approach to the IL setting, where the MDP is unknown. We assume the agent has access to sensors and for-purpose perception tools that are used to identify objects in the environment that correspond to propositions of the (unknown) logic specification (e.g., the variables of the formal language). Additionally, exclusively during training we assume a logic oracle that the agent can query to learn the high-level FSA state it is currently in. This assumption is more plausible and efficient to simulate compared to related works, which require knowledge of the full FSA (Paxton et al., 2017; Xie et al., 2018). Our model, the Logic-based Value Iteration Network (LVIN), learns the relevant part of the transition matrix (TM) describing the high-level FSA and directly integrates it into a differentiable recursive planning algorithm generalizing the Value Iteration

*equal contribution

Infer to Control: NIPS 2018 Workshop on Probabilistic Reinforcement Learning and Structured Control.
Network (VIN) proposed in (Tamar et al., 2016). To study the ability of LVIN to accurately recover the FSA transition matrix in the realizable setting, we define our tasks with LTL formulae and use shortest path algorithms to generate experts which optimally solve the tasks. We then use an imitation learning loss derived from these experts to train.

2 The Logic-based Value Iteration Network Model

Linear Temporal Logic

We use linear temporal logic (LTL) to formally specify the tasks we learn to solve (Clarke et al., 2001). Formulae $\phi$ constructed in LTL use the grammar

$$
\phi := p \mid \neg \phi \mid \phi_1 \lor \phi_2 \mid \diamond \phi \mid \phi_1 U \phi_2
$$

(1)

where $p$ is a proposition from a set of objects in the world (e.g., red light, car, and so on), $\neg$ is negation, $\lor$ is disjunction, $\diamond$ is “next”, and $U$ is “until”. The derived rules are conjunction ($\land$), implication ($\implies$), equivalence ($\iff$), “eventually” ($\diamond \phi = \text{True} U \phi$) and “always” ($\Box \phi = \neg \diamond \neg \phi$).

Intuitively, $\diamond \phi$ states that $\phi$ evaluates to true in the next world state, $\phi_1 U \phi_2$ means that $\phi_1$ is true until $\phi_2$ is true, $\diamond \phi$ means that there is a state where $\phi$ is true and $\Box \phi$ means that $\phi$ is always true. LTL formulae express logic-based constraints in an environment. LTL formulae can be converted into a finite-state automata (FSA); we use the software package SPOT (Duret-Lutz et al., 2016).

Inputs to the Model

LVIN receives the current state of the environment. Exclusively during training, LVIN receives the expert action and the logic oracle gives access to the current FSA state.

Training Loss

There are two training losses: (1) a cross-entropy loss for predicting the next FSA state, governing the learning of the FSA TM, and (2) a standard IL loss for predicting the next action.

The Model

Our model is based off of the VIN model (Tamar et al., 2016). The main differences between the original approach and ours is that we integrate the logic specification into the end-to-end differentiable setup by (1) adding an extra channel for logic propositions and FSA states in the input and (2) adding parameters in the form of a TM which depends on the propositions and FSA states. We use this approximate TM, denoted as $\overline{T}$, as a part of the learning procedure and learn it in an end-to-end manner while the planning network and policy are being learned. This model can be described as learning a 3D convolution tensor with a highly specific sparse structure consisting of $\overline{T}$ and $(row, col)$-action kernels $P_a$ over a product of VINs (one for each FSA state). The action kernels are shared across the VINs as well. For the $i^{th}$ VIN, $\overline{R}_i$ corresponds to the estimated input reward map, defined as in the VIN algorithm, and $\overline{V}_i$ corresponds to the estimated value map, which is updated by $\overline{V}'_i$ after each pass through the network. $\overline{Q}_i$ is the estimated Q-function for the $i^{th}$ VIN. Max-pooling over actions yields $\overline{V}_i$. The $\overline{V}_i$ are “wired together”, as stated in lines 13 - 14 of Alg. 1 so that there are connections between the parallel VINs. $P$ and $\overline{T}$ are the estimated action-kernels and TM. We update the network by backpropagating the imitation loss through all layers. To run LVIN after training, we only require the initial FSA state. We use the Q-function for the current FSA state to select the best action to take. Having selected this action, we can predict the next FSA state using $\overline{T}$, and use that as input in the next step. For an intuitive picture of LVIN, see Appendix A.

3 Experiments

3.1 Baselines

<table>
<thead>
<tr>
<th></th>
<th>LVIN</th>
<th>Hard-coded LVIN</th>
<th>CNN</th>
<th>VIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kitchen Domain</td>
<td>99.84%</td>
<td>99.76%</td>
<td>99.20%</td>
<td>38.92%</td>
</tr>
<tr>
<td>Longterm Domain</td>
<td>100.00%</td>
<td>N/A</td>
<td>82.80%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Driving Domain</td>
<td>99.60%</td>
<td>98.40%</td>
<td>8.60%</td>
<td>99.90%</td>
</tr>
</tbody>
</table>

(a)

Table 1

VIN: We compare the performance of LVIN to VIN. We do not attempt to predict the next FSA state, nor do we incorporate a TM into the learning process.
Algorithm 1 LVIN Training

1: procedure LVIN-TRAINING
2: Inputs: fsa, row, col, proposition map
3: Inputs (training): expert action, next fsa
4: To learn: Transition matrix \( \mathbf{T} \in \mathbb{R}^{fsa \times fsa \times props} \)
5: Low-level action kernels \( \bar{P}_a \)
6: Build the model:
7: Create a VIN for each FSA state.
8: Share \( P_a \) across each VIN.
9: Learn \( \mathbf{T} \) by predicting next fsa given fsa, given expert action.
10: Normalize \( \mathbf{T} \) so that it is row-stochastic.
11: Wire together the outputs of VINs with the learned \( \mathbf{T} \):
12: for all FSA states \( s, s' \) s.t. \( \mathbf{T}[s, \text{row}, \text{col} \rightarrow \text{prop at (row, col)} (s', \text{row}, \text{col})] > 0 \) do
13: \( V_s(\text{row}, \text{col}) := \mathbb{E}_{s' \sim \mathbf{T}}[V_{s'}(\text{row}, \text{col})] \)
14: end for
15: Backpropagate the imitation loss through the model.
16: end procedure

(a) The 8 x 8 kitchen domain. (b) The 12 x 9 longterm domain. (c) The 14 x 14 driving domain.

Figure 1

Hard-coded LVIN: It is not necessary to learn the TM from data – if the FSA is known, then a valid TM can be constructed from the FSA. We compare the performance of LVIN to LVIN with a hard-coded TM to see if learning the TM degrades performance.

CNN: We formulate a less constrained version of LVIN that uses a 3D CNN instead of a TM to transfer values between FSA states. The CNN operation acts on a concatenation of the proposition matrix and the value function, returning the next iteration of the value function.

3.2 Example Environments

Kitchen Domain The kitchen domain is an 8 x 8 grid with deterministic actions which enable movement to adjacent cells. The domain is a kitchen with three propositions: \( o \) for obstacle, \( m \) for milk, and \( c \) for cereal. The agent, a robot with a bowl, should first fill the bowl with milk (visit \( m \)) and then put in the cereal (visit \( c \)) while avoiding randomly placed obstacles (chairs, tables, and plants).

Longterm Domain The longterm domain is a 12 x 9 grid environment with the goal of showcasing LVIN’s ability to learn relatively complex sequential specifications. In this environment, shown in Fig. 1b, there are five rooms, four doors, four keys, and a goal. In order to progress to the goal, the agent must learn to make a “longterm” plan – it must first pick up Key A, then go get Key D, then Key B, then Key C, before it can finally access the room in which the goal is located.

Driving Domain The driving domain (Fig. 1c) is a 14 x 14 grid environment with the goal of showcasing LVIN’s ability to learn and encode rules, in this case three “rules of the road.” The model must learn three rules of the road: avoid obstacles, prefer the right lane over the left lane, and stop at red lights.

The results are shown in Table 1a.
Fixing Expert Errors  Our model can also be used to fix the mistakes of faulty experts. Suppose the real-world driving data contains behavior from drivers running red lights. We model this scenario in Table 5 where the Unsafe TM shows a scenario in which the model has learned to run a red light 10% of the time. We correct the TM by setting the initial state entry to 0 so that the model learns to never run a red light. The results are shown in Table 10 along with the performance of the LVIN and CNN models trained on data generated from $\phi_k$. The tests highlight a shortcoming of the CNN model: Since the TM is not incorporated into the planning step, modifying the TM does not change the behavior of the agent.

Table 3: Rollout performance of the Unsafe and Safe TMs.

4 Conclusion and Future Work

Developing interpretable and manipulable models that learn to plan is an ongoing goal in deep policy learning. This work demonstrated this goal can be achieved using logic automata. By learning an FSA transition matrix in conjunction with a planning module, we were able to build a model that a human can control intuitively.
References


A  Diagrams of the LVIN Model

Figure 2: The FSA transition matrix (learned by predicting the next FSA state) “wires” together the
value maps of each FSA state (S0, S1, G, T). Each proposition (A: first goal, O: obstacle, B: second
goal) is associated with a row of the learned TM for each FSA state, which induces the wires between
FSA states. The value of a “wired” cell is the expected value (w.r.t. TM) of cells the outgoing wires
point to.
Figure 3: LVIN forward pass: For each FSA state, we have a value map \((row, col)\)-action kernels \(P_a((row', col')|(row, col))\) are shared across FSA states, and applied to produce Q-maps. Max-pooling yields updated value maps for each FSA state. The “wired” output is depicted in detail in Fig. 2. The process is looped \(k\) times.