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

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 ϕ constructed in LTL use the grammar

$$\phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2 \tag{1}$$

where p is a *proposition* from a set of objects in the world (e.g., red light, car, and so on), $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", and $\mathcal{U}$ is "until". The derived rules are conjunction ($\wedge$), implication ($\implies$), equivalence ($\leftrightarrow$), "eventually" ($\Diamond\phi = \texttt{True}\,\mathcal{U}\,\phi$) and "always" ($\Box\phi = \neg\Diamond\neg\phi$). Intuitively, $\bigcirc\phi$ states that ϕ evalutes to true in the next world state, $\phi_1 \mathcal{U} \phi_2$ means that ϕ_1 is true until ϕ_2 is true, $\Diamond\phi$ means that there is a state where ϕ is true and $\Box\phi$ means that ϕ 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 $\overline{\texttt{TM}}$, 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 $\overline{\texttt{TM}}$ and ($\texttt{row}$, $\texttt{col}$)-action kernels $\bar{P}_a$ 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, $\bar{R}_i$ corresponds to the estimated input reward map, defined as in the VIN algorithm, and $\bar{V}_i$ corresponds to the estimated value map, which is updated by $\bar{V}'_i$ after each pass through the network. $\bar{Q}_i$ is the estimated Q-function for the i^{th} VIN. Max-pooling over actions yields $\tilde{V}_i$. The $\tilde{V}_i$ are "wired together", as stated in lines $13-14$ of Alg. 1, so that there are connections between the parallel VINs. $\bar{P}$ and $\overline{\texttt{TM}}$ 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 $\overline{\texttt{TM}}$, and use that as input in the next step. For an intuitive picture of LVIN, see Appendix A.

3 Experiments

3.1 Baselines

	LVIN	Hard-coded LVIN	CNN	VIN
Kitchen Domain	99.84%	99.76%	99.20%	38.92%
Longterm Domain	100.00%	N/A	82.80%	0.00%
Driving Domain	99.60%	98.40%	8.60%	99.90%

(a)

	LVIN	CNN	Mod. LVIN	Mod. CNN
ϕ_{k1}	99.84%	99.76%	N/A	N/A
ϕ_{k2}	99.70%	99.30%	99.80%	6.30%
ϕ_{k3}	100.00%	92.90%	97.80%	1.10%

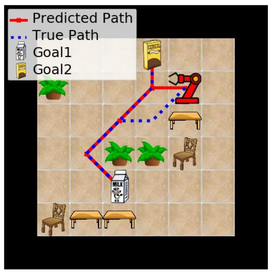
(b)

Table 1

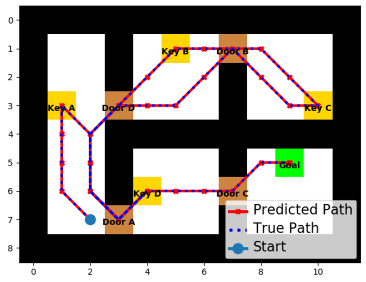
VIN: We compare the performance of LVIN to VIN. We do not attempt to predict the next FSA state, nor do we incorporate a TM into the learning process.

Algorithm 1 LVIN Training

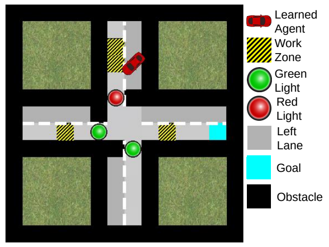
```
 1: procedure LVIN-TRAINING
 2:     Inputs: fsa, row, col, proposition map
 3:     Inputs (training): expert action, next fsa
 4:     To learn:
 5:         Transition matrix $\overline{\texttt{TM}} \in \mathbb{R}^{\texttt{fsa}\times\texttt{fsa}\times\texttt{props}}$
 6:         Low-level action kernels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



(a) The 8×8 kitchen domain.



(b) The 12×9 longterm domain.



(c) The 14×14 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 8×8 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 12×9 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 14×14 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.

I	da	db	dc	dd	g	ka	kb	kc	kd	o	e
I					1						1
G											
S1						1					
S2											
S3											
S4			1								
T	1	1		1			1	1	1	1	

(a) Initial state TM

S1	dd	kd
I		
G		
S1		
S2		
S3		
S4		1
T	1	

(b) $S1$

S2	g
I	
G	1
S1	
S2	
S3	
S4	
T	

(c) $S2$

S3	dc	kc
I		
G		
S1		
S2		1
S3		
S4		
T	1	

(d) $S3$

S4	db	kb
I	1	
G		
S1		
S2		
S3		1
S4		
T		

(e) $S4$

Table 2: The learned TM of the longterm domain. Cells of interest are highlighted in yellow. $da, db, \ldots$ correspond to Door A, Door B, etc. $ka, kb, \ldots$ correspond to Key A, Key B, etc. g corresponds to goal, o to obstacle, and e to empty space (i.e. no proposition).

3.3 Demonstrating Interpretability and Manipulability

Interpretability One benefit of learning the TM is that the values of the TM have meaningful interpretations. Table 2 contains parts of the TM of the longterm domain learned by LVIN. If the agent is in the initial state (Table 2a) and enters proposition ka (it picks up Key A), then the predicted next FSA state is $S1$. In the TM of the initial state, Table 2a, we see that all doors and keys map to the trap state, except for Key A, which maps to state $S1$ – the model has learned that when in the initial state, the agent cannot travel through doors and that it must pick up Key A before any other key. Partial TMs for the other states show that the model has learned a sequence of keys to pick up, and that it cannot pass through the door associated with its key until it has picked up the key. Unexpected transitions are highlighted in red. In every case, unexpected transitions occur where the model has not actually observed a transition (columns highlighted in grey) but rather had to infer the value.

Manipulability Since we can interpret the TM, we can also modify it to change the behavior of the agent. To demonstrate, we will manipulate the TM that was learned in the kitchen domain. The learned TM defines spec ϕ_{k1}: first add milk, then cereal. However, the owner of a cereal-preparing robot may want to tell the robot to do ϕ_{k2}: add first cereal, then milk.
The modifications to the TM to achieve ϕ_{k2} are shown in Table 3a. To make the agent go first to c, we modify the initial state's TM so that m maps to the initial state and c maps to $S1$. To the agent, this means that going to m does nothing, whereas going to c will bring it to the next state. We then modify the TM for $S1$ so that m maps to the goal state and c maps back to $S1$. The results are shown in Table 1b, along with the performance of the LVIN and CNN models trained on data generated from ϕ_{k2}. 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.

I	m	c	o	e
I	1	-		1
S1	-	1		
G				
T			1	

S1	m	c	o	e
I			1	
S1	-	1		1
G	1	-		
T				

(a) $\boldsymbol{\phi_{k1} \rightarrow \phi_{k2}}$

	Unsafe	**Safe**
Initial State	red light	red light
Initial	0.1	0.0
Left Lane	0.0	0.0
Goal	0.0	0.0
Red Light	0.9	1.0
Trap	0.0	0.0

Rollout Performance	
Unsafe TM	9.88%
Safe TM	0.00%

Table 3

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 3, 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 and the red light state entry to 1. We perform 1000 rollouts using each of these TMs. The Unsafe TM results in the agent running 9.88% of red lights while the Safe TM prevents the agent from running any red lights.

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

P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” *ICML ’04 Proceedings of the twenty-first international conference on Machine learning*, p. 1, 2004.

H. Daumé III, J. Langford, and D. Marcu, “Search-based structured prediction,” *Journal of Machine Learning*, vol. 75, pp. 297–325, 2009.

S. Ross, G. Gordon, and J. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 627–635, 2011.

Y. Yue and H. Le, “Imitation learning tutorial,” Tutorial at ICML 2018, 2018. [Online]. Available: https://sites.google.com/view/icml2018-imitation-learning/home

C.-I. Vasile, K. Leahy, E. Cristofalo, A. Jones, M. Schwager, and C. Belta, “Control in belief space with temporal logic specifications,” in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 7419–7424.

C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, “Minimum-violation scltl motion planning for mobility-on-demand,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1481–1488.

C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov, “Combining neural networks and tree search for task and motion planning in challenging environments,” *ArXiv e-prints*, 2017.

S. Xie, A. Galashov, S. Liu, S. Hou, R. Pascanu, N. Heess, and Y. W. Teh, “Transferring task goals via hierarchical reinforcement learning,” 2018.

A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, “Value iteration networks,” in *Advances in Neural Information Processing Systems 29*, 2016, pp. 2154–2162.

E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 2001.

A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, “Spot 2.0 — a framework for LTL and ω-automata manipulation,” in *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA’16)*, ser. Lecture Notes in Computer Science, vol. 9938. Springer, Oct. 2016, pp. 122–129.

A Diagrams of the LVIN Model

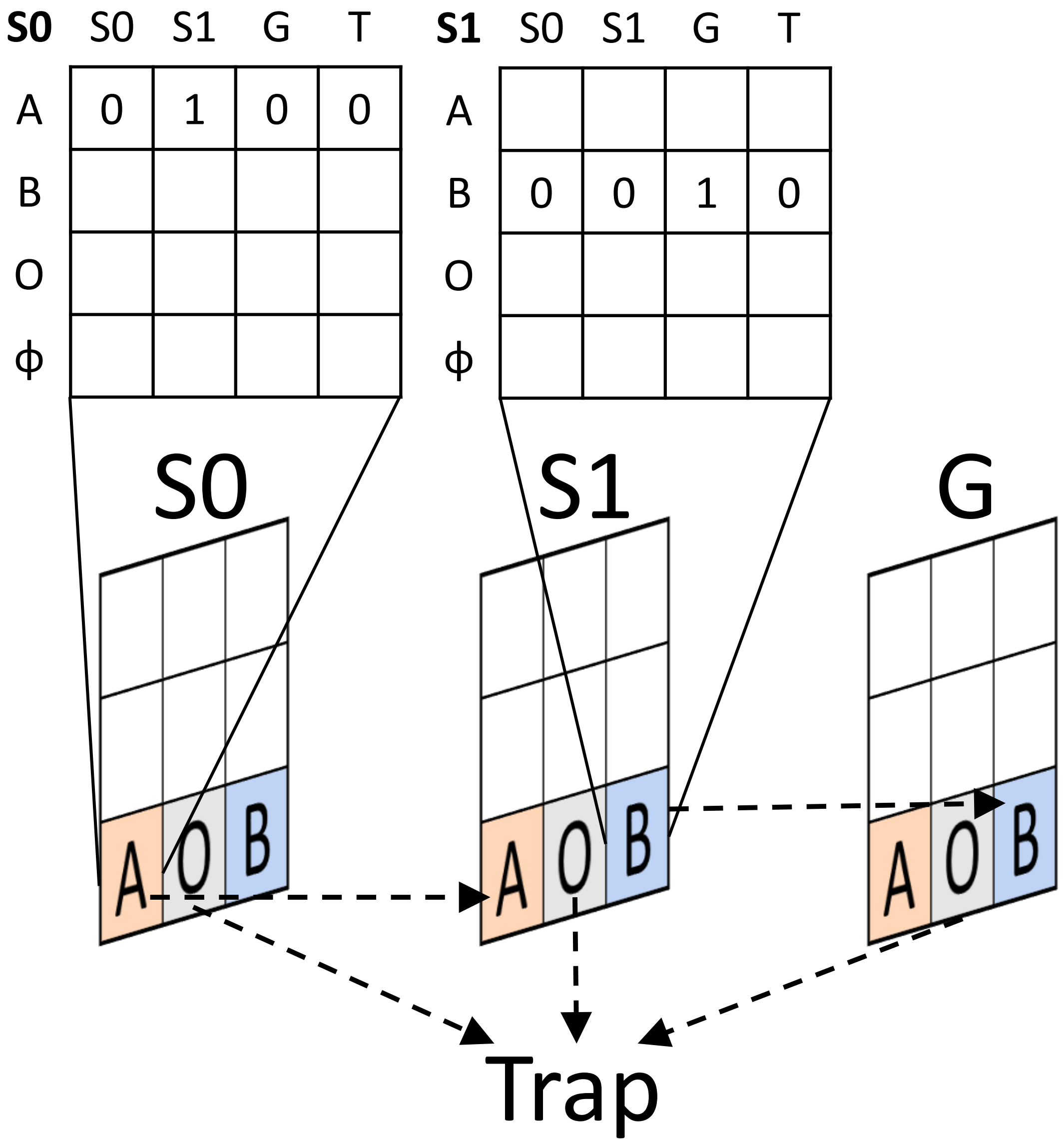


Figure 2: The FSA transition matrix (learned by predicting the next FSA state) "wires" together the value maps of each FSA state (S0, S1, G, T). Each proposition (A: first goal, O: obstacle, B: second goal) is associated with a row of the learned TM for each FSA state, which induces the wires between FSA states. The value of a "wired" cell is the expected value (w.r.t. TM) of cells the outgoing wires point to.

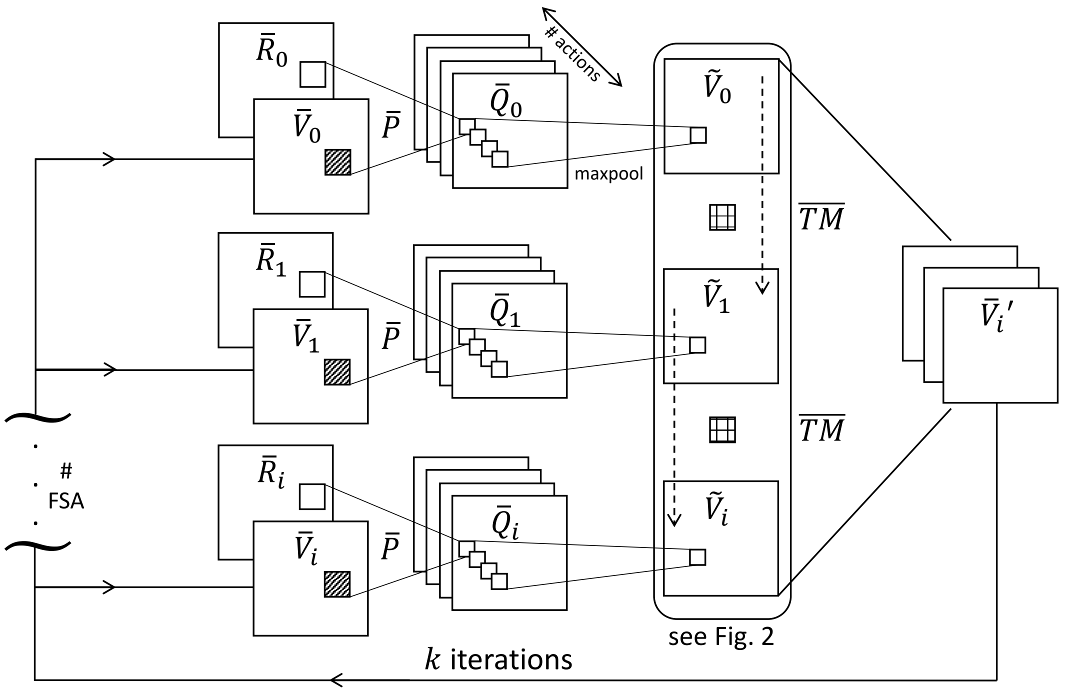


Figure 3: LVIN forward pass: For each FSA state, we have a value map. (`row, col`)-action kernels $\bar{P}_a((\texttt{row', col'})|(\texttt{row, col}))$ are shared across FSA states, and applied to produce Q-maps. Max-pooling yields updated value maps for each FSA state. The "wired" output is depicted in detail in Fig. 2. The process is looped k times.