Advances in Robotic Learning Paper Summary: Learning to Plan with Logical Automata

Brandon Araki[1,*], Kiran Vodrahalli [2,*], Thomas Leech[1,3], Cristian-Ioan Vasile[1], Mark Donahue[3], Daniela Rus [1]
Presented by: Frank Liu, Evan Lam, Michael Drolet

Abstract— **This electronic document is a live template. The various components of your paper [title, text, heads, etc.] are already defined on the style sheet, as illustrated by the portions given in this document.**

I. INTRODUCTION

Recall the daunting time when you had to learn how to drive for the first time. In order to drive properly, you not only need to learn how to drive a car, but also learn the rules of the road. To learn how to drive a car, many of us practiced driving and learning all the mechanics to make the car move. Most of us went to a driving school, where a driving instructor taught us certain driving rules in the United States. Others might watch instructional videos from experts online. One way or another, we developed a mental model of the rules of the road through imitating an expert.

Now there are two parts to this learning. The first part is learning the lower level actions in order to operate and drive a car. The second part is developing a mental model or representation of an interpretable policy, such as the rules of the road. The structure of the learned policy should be grounded in meaningful interpretations.

When learning the rules of the road, a naive assumption is that all experts have taught properly and that all the instruction received is correct. If there are bad or even illegal driving habits, they will need to be corrected to ensure safe driving. For example, if a person runs a red light or makes an illegal u-turn, a police officer would reprimand and correct that behavior by issuing a ticket. We are able to be corrected because the rules in our heads are manipulable. Consequently, a human operator can easily modify a learned policy to perform similar but different policies.

Applying this to robot learning, the authors work toward teaching a robot to learn from demonstrations using not only a low-level policy, but also a high level policy that is interpretable and manipulable. The authors create a Logic-based Value Iteration Network (LVIN) which utilizes these two principles in learning policies. The policies that a robot learns should be interpretable, where there is a set of learned representation of rules. The behavior of the robot should be manipulable, where the rules can be changed in a predictable way which results in changed behavior. The LVIN model is a recurrent, convolutional neural network that uses value iteration over a learned Markov Decision Process (MDP). This MDP factors into two separate parts: a Finite State Automaton (FSA) corresponding to the low-level policy, as well as a bigger MDP corresponding to the rules in an environment.

A major benefit to using this approach for learning is that a robot can both learn from demonstration and modify the learned policy to be safe. In the case of the driving example, if a robot was learning the rules of the road and five percent of the training data included illegal left turns (resulting in crashes), then the robot would learn the policy which crashes five percent of the time. With the author's approach in robotic learning, such a policy can be corrected to prevent crashes. These rules can also be applied in many alternative scenarios.

In this paper, the authors main contributions are

1) A Logic-based Value Iteration Network (LVIN) model that learns policies for robot learning from an imitation learning perspective. The authors show the effectiveness of the LVIN model through four different benchmark scenarios.
2) Presenting that the model can learn transitions from state to state, therefore showing that it can interpret the rules.
3) Demonstrating that the learning framework is manipulable, thus generalizable to other tasks and able to fix mistakes without extra training or experts.

II. RELATED WORK

A. Logic-based Approaches

Linear Temporal Logic (LTL) is a temporal logic language that is used to specify complex tasks which can then be translated into a finite state automaton (FSA). One work published in 2017 uses LTL to define the constraints on a Monte Carlo Tree Search for generating task plans to navigate through complex environments [1]. Other works have used LTL-derived FSAs in conjunction with MDPs to make reinforcement learning more efficient [2, 3].

B. Multitask and Meta Learning

Multitask learning methods can refer to methods used to learn in a multitask setting, or when the model is learning several things at the same time. A paper published in 2017 described a framework for multitask deem reinforcement learning that is guided by policy sketches. The policy sketches are symbolic representations of a task that describe

[1]MIT CSAIL, Cambridge, MA 02139
[2]Columbia University, New York City, NY 10027
[2]MIT Lincoln Laboratory, Lexington, MA 02421
[*]Authors contributed equally

its component parts and subtasks [4]. A paper published in 2016 that proposes the use of a recurrent neural network (RNN) to represent a ”fast” reinforcement learning algorithm which falls under the second type of multitask learning mentioned. In the proposed framework both the agent and the learning process of the agent itself are optimized throughout the learning process. This framework allows the learned agent to adapt to a task at the time of deployment via meta-learning [5].

The goal of meta-learning is to train a model on various learning tasks such that it con solve new learning tasks with a small number of training samples. Meta-learning has been studied and applied to various computing disciplines including the fast adaptation of deep networks [6] and one-shot learning in which robots learn how to complete a new, unseen task after only a single visual demonstration [7, 8].

C. *Faulty Experts*

In imitation learning, the goal is to mimic the demonstration of an expert. However, faulty experts can result in the wrong behavior being learned and replicated. To combat the effects of faulty experts, a paper published in 2015 introduces a new learning paradigm that combines imitation learning with intention learning, which learns the expert’s intent rather than strictly copying its actions [9]. Another work published in 2019 proposed a unified reinforcement learning algorithm that normalizes the Q-function and learning through demonstration and interactive refinement in the environment using the same objective which helps protect from faulty demonstrations by not forcing the method to mimic all of the examples in the dataset [10].

D. *Hierarchical Learning*

The LVIN model introduced by the authors is an instance of hierarchical learning which was first introduced in a paper by Parr and Russel in 1998 which introduced the approach that allows for the use of prior knowledge to reduce the search space [11]. The options framework outlined by Sutton, Precup, and Singh enables the abstraction of temporal knowledge to be used in specifying policies [12]. One recent work utilize the hierarchical learning and options frameworks to speed up learning and reduce expert effort and cost of exploration through a framework that combines imitation learning and reinforcement learning at different levels [13].

III. Problem Statement

The overall goal is of this paper is to create a model that learns from demonstration a low-level policy, as well as a high level policy that is interpretable and manipulable. For our problem statement, we make a few assumptions.

- We assume that rules can be encoded as finite state automaton (FSA).
- We assume that the relative features in an environment can be detected.
- We assume that the FSA states are known.
- We assume that the environment outputs current FSA state and low level state at each time step.
- The learned policy comes from the learned transitions among the FSA states and low-level transitions.

When authors say that they can detect the relative features of an environment, what they mean is that these features can be treated as logical propositions or a true/false variable. For example in a 2D grid environment, if there is a red light in the environment then at that particular grid position (x,y) where the red light is located the variable would be set to true. If there was no red light, then the variable would be set to false.

We assume that the expert is following some sort of finite state controller. This finite state maps to the variables in the environment with different states corresponding to the variables environment. There are different actions based on the variable. For example, move forward on a green light as an action in the state machine. The FSA which the expert follows is the overall policy in which our network wants to learn. The learnt FSA is interpretable because the model can learn expert trajectories. Additionally, this model can obtain new policies without re-learning a changed expert FSA.

A. *Value Iteration Network*

The LVIN network is based off of the Value Iteration Network [15]. The Value Iteration Network architecture is a fully differentiable version of value iteration. The Q function is calculated using a convolution and the Value function is calculated using a max pool function. This lets you learn the reward function and transition function. One limitation is that this Value Iteration Network is that it works best in a 2D grid environment due to the structure of the network.

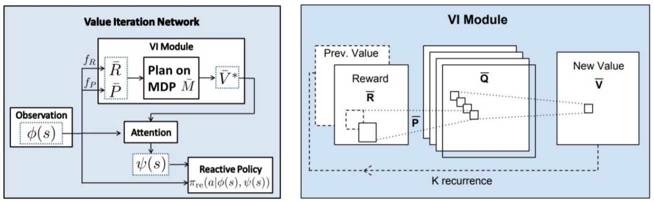


Fig. 1. A block diagram of the Value Iteration Network. The module is shown on the right. The reward function is R, while the transition function is P.

B. Logic-Based Value Iteration Network

There are two differences between the Logic-Based Value Iteration Network and the Value Iteration Network.

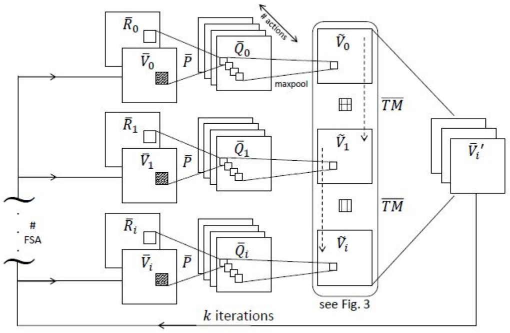


Fig. 2. A block diagram of the Logic-Based Value Iteration Network, figure 4 in the paper [14].

The first difference is that LVIN has a Value Iteration Network for every FSA state. As a result, the reward function and transition function are learned for every FSA state.

The second difference is there is a second convolution which is applied afterwards, which is represented by TM. This convolution is based off the transitions of the FSA. The second convolution helps the model learn the transitions of the FSA's.

IV. Experiments

The LVIN model was tested against 2-3 baselines in 4 different virtual domains (Kitchen, Longterm, Pickworld, and Driving). Each domain emphasized a key claim of the LVIN network's capabilities.

A. Data Generation

The authors use a software package to convert specified tasks into FSAs. Each FSA state contains a goal state, as well as undesired termination states. For each of the four domains, the authors generate proposition variables. The expert trajectories were created by using the Dijkstra's shortest path algorithm for the imitation learning data.

B. Base Lines

- Value Iteration Network. The authors compared the LVIN to the VIN.
- Hard-coded LVIN. The TM is hard coded into the LVIN. The LVIN with a learned TM is compared to a Hard-coded LVIN with a known TM.
- CNN. Instead of a second convolution for the learned TM, there is a 3-D convolutional neural network attached at the end that learns the transitions of the FSAs.

C. Domains

Kitchen. In a 8x8 grid world in figure 3, the kitchen domain had milk, cereal and obstacles in the environment. The goal was to first fill a bowl with milk before filling it with cereal while avoiding the obstacles. Measuring the LVIN model against the baselines, we see in figure 4 that the LVIN model performs just as well as the hard-coded LVIN and CNN while the VIN performed poorly.

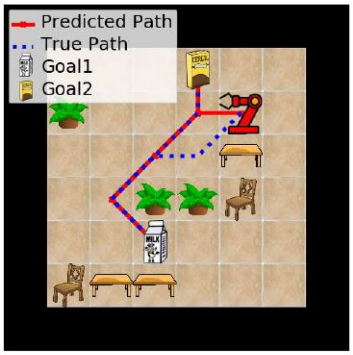


Fig. 3. Kitchen Domain

Kitchen Domain

	LVIN	Hard-coded LVIN	CNN	VIN
Action Accuracy	98.85%	99.07%	98.29%	68.05%
FSA Accuracy	99.71%	99.73%	99.73%	N/A
Performance over 5000 rollouts				
Both Goals, Correct Order	99.84%	99.76%	99.20%	38.92%
Only Milk	0.02%	0.06%	0.00%	19.88%
Only Cereal	0.02%	0.00%	0.00%	34.10%
Both Goals, Wrong Order	0.02%	0.00%	0.74%	5.28%
No Goal	0.10%	0.18%	0.06%	1.82%

Fig. 4. Performance in the Kitchen domain

Longterm. In the 12x9 grid world in figure 5, the goal is to pick up each key to access the corresponding door to reach the goal position. This emphasizes learning long term complex behavior to reach the goal.

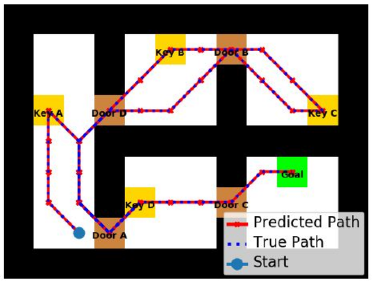
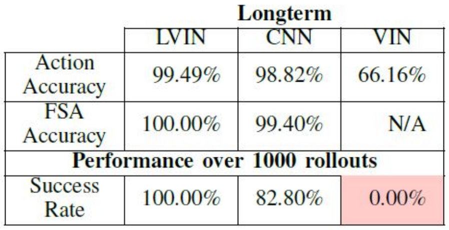


Fig. 5. Longterm Domain

We see that the LVIN and CNN have good accuracy while the VIN struggles to learn the complex paths.



Longterm

	LVIN	CNN	VIN
Action Accuracy	99.49%	98.82%	66.16%
FSA Accuracy	100.00%	99.40%	N/A
Performance over 1000 rollouts			
Success Rate	100.00%	82.80%	0.00%

Fig. 6. Performance in the Longterm domain

Pickworld. In a 18x7 grid world in figure 7, the goal in the Pickworld domain is to first pick up either a sandwich a or a burger b and put it in a lunchbox d, and then pick up a banana c and put it in the lunchbox d. The LVIN, the Hard-coded LVIN and the CNN performed the task quite well. The VIN struggled to learn the task. The LVIN models were the most successful.

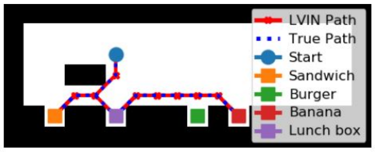
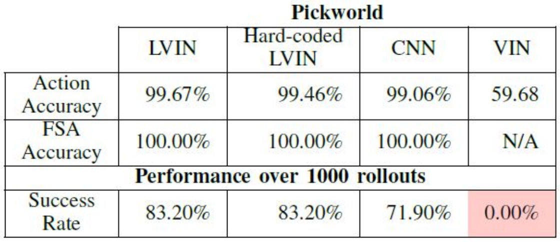


Fig. 7. Pickworld Domain



Pickworld

	LVIN	Hard-coded LVIN	CNN	VIN
Action Accuracy	99.67%	99.46%	99.06%	59.68
FSA Accuracy	100.00%	100.00%	100.00%	N/A
Performance over 1000 rollouts				
Success Rate	83.20%	83.20%	71.90%	0.00%

Fig. 8. Performance in the Pickworld Domain

Driving. In a 14x14 gridworld, this driving showcases the LVIN's ability to learn the rules of the world. The goal was to reach the goal position while learning to stop on red lights and wait until it turned green and to avoid obstacles in the environment. Surprisingly, the VIN does better than the other baselines probably because there is only one sequential goal. LVIN, Hard-coded LVIN and CNN perform quite well.

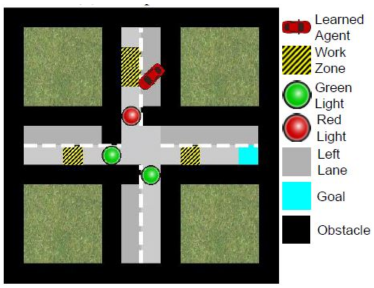
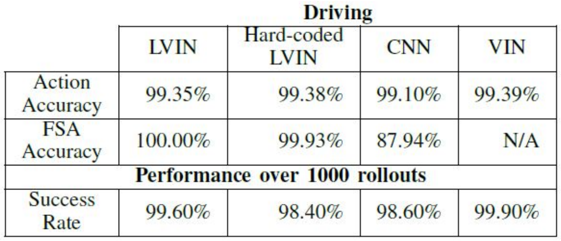


Fig. 9. Driving Domain



Driving

	LVIN	Hard-coded LVIN	CNN	VIN
Action Accuracy	99.35%	99.38%	99.10%	99.39%
FSA Accuracy	100.00%	99.93%	87.94%	N/A
Performance over 1000 rollouts				
Success Rate	99.60%	98.40%	98.60%	99.90%

Fig. 10. Performance in the Driving Domain

D. Interpretability and Manipulability Analysis

Interpretability. Interpretability means that there is a meaningful learned FSA representations of rules. The learned TM should be similar to that of the true expert trajectory. We examine the different states and the corresponding learned TM and compare that with the true TM. Overall we see that the learned TM is similar to the true TM.

Kitchen. For the kitchen, the goal of S0 is to reach goal a and S1 is to reach goal b.

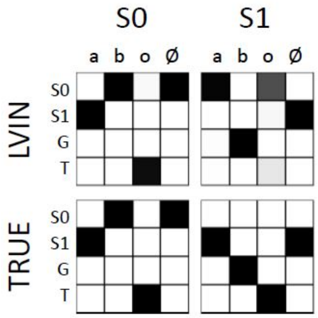


Fig. 11. Kitchen Transition Matrix

Pickworld. For the Pickworld, the goal of S0 is to reach either a or b. The goal of S1 is to reach d. The goal of S2 is to reach c. The goal of S3 is to reach D again. The unexpected transitions are highlighted in red.

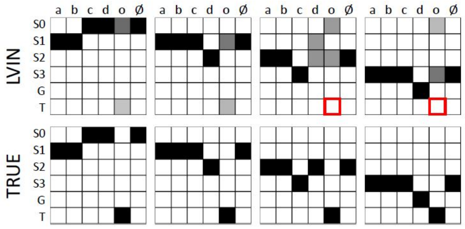


Fig. 12. Pick world Transition Matrix

Driving. For the Drive world, the S0 is when the car drives on the right lane to reach the goal. S1 is when the car drives on the left lane to reach the goal. S2 is when the care is at a red light. The unexpected transitions are highlighted in red.

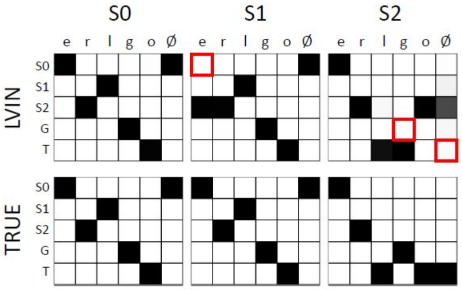


Fig. 13. Drive world Transition Matrix

Longterm. For the Longterm world, there are far more states available compared to the other domains. There is a total of 10 propositions which map to 33 different states. One rule that confines the state space is that all the keys must be picked up in order as a result only 6 of the 33 possible states are visited. Each table is associated with a single state and shows how to get to the next FSA state.

S0	da	db	dc	dd	ka	kb	kc	kd	g	o	ϕ
S0									1		1
S1					1						
S2			1								
S3											
S4											
G											
T	1	1		1		1	1	1		1	

(a) In the initial state, Door A is not allowed, and Key A leads to state $S1$

S1	dd	kd
S0		
S1		
S2		1
S3		
S4		
G		
T	1	

(b) In $S1$, Key D leads to $S2$.

S2	db	kb
S0	1	
S1		
S2		
S3		1
S4		
G		
T		

(c) In $S2$, Key B leads to $S3$.

S3	dc	kc
S0		
S1		
S4		
S3		
S2		1
G		
T	1	

(d) In $S3$, Key C leads to $S4$.

S4	g
S0	
S1	
S2	
S3	
S4	
G	1
T	

(e) $S4$ leads to goal.

Fig. 14. Transition Matrix of the Long Term domain. The unvisited states are gray. The unexpected transitions are red. The cells of interest are in yellow.

Manipulability. Manipulability means that the behavior can be changed when the rules are modified in a predictable way which modifies the FSA transitions. The authors implement the pickworld domain onto a jaco arm to show the LVIN model working in real life. The objects were tracked with an optitracker and commands were sent via ROS. The transition matrix is modified for new behavior without relearning.

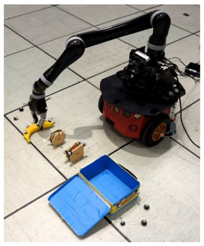
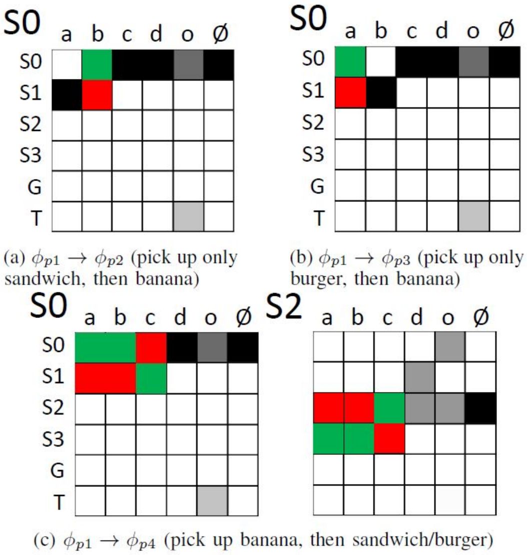


Fig. 15. Pickworld with a Jacoarm.

Below is the modified transition matrix with red representing deleted values and green representing added values. The authors use three different changed transition matrices to test.



(a) $\phi_{p1} \rightarrow \phi_{p2}$ (pick up only sandwich, then banana)

(b) $\phi_{p1} \rightarrow \phi_{p3}$ (pick up only burger, then banana)

(c) $\phi_{p1} \rightarrow \phi_{p4}$ (pick up banana, then sandwich/burger)

Fig. 16. Modified transition matrix.

We saw that the performance of the LVIN, Mod. LVIN and the CNN were largely successful. The LVIN and CNN are baselines directly trained on the new specifications. The modified LVIN is the old LVIN with a changed transition matrix. The modified CNN did poorly because the TM is not included and the CNN performs the old specification. This highlights the need for manipulability which the LVIN is capable of. The failures resulted from the banana falling out the the jaco arm grip.

		LVIN	CNN	Mod. LVIN	Mod. CNN
Performance over 1000 rollouts					
ϕ_{p2}	Sandwich-to-Burger Ratio	1.0	1.0	1.0	0.58
ϕ_{p3}	Burger-to-Sandwich Ratio	1.0	1.0	1.0	0.43
ϕ_{p4}	Success Rate	89.80%	90.60%	95.00%	1.10%

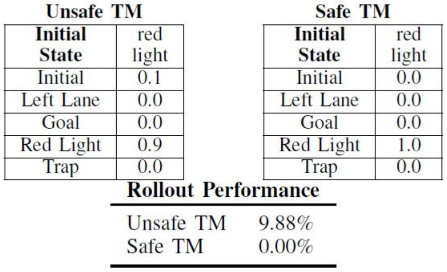
Fig. 17. Jaco arm performance.

	ϕ_{p1}	ϕ_{p2}	ϕ_{p3}	ϕ_{p4}
Success Rate	18/20	19/20	18/20	20/20
Failure Modes	2/20 banana slipped out of hand	1/20 banana slipped out of hand	1/20 banana slipped out of hand 1/20 bad path	N/A

Fig. 18. Reason for failure.

V. Fixing Expert Mistakes

Sometimes the data that is learned has a mistake. In this case, the authors used the driving world and had an issue of running a red light 10 percent of the time. The authors corrected this by setting the initial state entry to 0 and red light state entry to 1 to show there is a red light there.



Unsafe TM

Initial State	red light
Initial	0.1
Left Lane	0.0
Goal	0.0
Red Light	0.9
Trap	0.0

Safe TM

Initial State	red light
Initial	0.0
Left Lane	0.0
Goal	0.0
Red Light	1.0
Trap	0.0

Rollout Performance	
Unsafe TM	9.88%
Safe TM	0.00%

Fig. 19. Safety driving scenario.

VI. Discussion and Analysis

The authors mentioned was that the LVIN model works for finite grid world environments. These finite grid world environments are limited to a 2D grid. These applications might be incredibly useful integrated into a factory workplace with robots in highly controlled environments.

However, the real world is three dimensional and highly unpredictable. These finite grid world environment limits LVIN's real world use case. While the authors have a real world experiment with the Jaco arm to show the LVIN model's effectiveness, the real world experiment exists in a highly controlled environment which is not representative of real world situations.

A logical next step could be extending the 2D grid into a 3D grid world. We would imagine that the larger MDP for the rules of the world would add another dimension and the smaller FSA would add more states. Overall the new network model would be more complex. While the complexity of the network would increase, we do not believe the complexity change would not be too big of a problem in training on modern computers which train significantly large machine learning models.

It also seems the manipulability of the learning is limited. The authors modify the state after detecting an error and to correct bad behavior. It would be quite interesting to see if the model can automatically detect errors and self correct. We imagine there can be neural network infrastructure which can be added for error detection.

It would be quite interesting to explore the LVIN model with moving objects/obstacles in the 2D grid environment. How much would that effect the learned policies and how much impact would introducing such dynamic objects change the output behavior.

VII. CONCLUSIONS

In conclusion, the authors introduce a Logic-Based Value Iteration network which can learn policies from imitation learning and demonstration. The authors tackle how to generalize a learned policy for a particular behavior to a larger set of tasks. Additionally, the authors address how to deal with incorrectly learned policies from incorrect demonstration. This network is a combination of a finite state automaton and a larger markov decision process. The LVIN network is a generalization of the Value Iteration Network, where the LVIN network learns the relevant transitions and creates a policy from the transitions of the FSA's. The key idea of the LVIN network is that a value iteration module is added to the end of a FSA and these modules get linked together.

The authors measure the LVIN network performance in four different virtual domains (Kitchen, Longterm, Pickworld, and Driving) and a real world implementation with a jaco arm. The LVIN model is shown to be accurate and effective in generalizing to new task specifications, and correcting errors. Future works can focus on expanding the model dimensionality for 3D scenarios and even self learn errors and dynamic objects in the world.

References

[1] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. ArXiv e-prints, 2017.

[2] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. arXiv preprint arXiv:1801.08099, 2018.

[3] Xiao Li, Yao Ma, and Calin Belta. Automata guided hierarchical reinforcement learning for zero-shot skill composition. ArXiv e-prints, 2017.

[4] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. ArXiv e-prints, 2016.

[5] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. ArXiv e-prints, 2016.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Modelagnostic meta-learning for fast adaptation of deep networks. Proceedings of the 34th International Conference on Machine Learning, 2017.

[7] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. arXiv preprint arXiv:1709.04905, 2017.

[8] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Fei-Fei Li, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. ArXiv e-prints, 2018.

[9] James MacGlashan and Michael L. Littman. Between imitation and intention learning. IJCAI'15 Proceedings of the 24th International Conference on Artificial Intelligence, pages 3692–3698, 2015.

[10] Yang Gao, Huazhe (Harry) Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. Proceedings of the 35th International Conference on Machine Learning, 2018.

[11] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In Advances in neural information processing systems, pages 1043–1049, 1998.

[12] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: Learning, planning, and representing knowledge at multiple temporal scales. Journal of Artificial Intelligence Research, 1:1–39, 1998.

[13] Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. ArXiv e-prints, 2018.

[14] Araki, Brandon & Vodrahalli, Kiran & Leech, Thomas & Vasile, Cristian-Ioan & Donahue, Mark & Rus, Daniela. (2019). Learning to Plan with Logical Automata.

[15] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In Advances in Neural Information Processing Systems 29, pages 2154– 2162, 2016.