Learning, Reasoning, and Planning with Relational and Temporal Neural Networks

by

Jiayuan Mao

B.Eng., Tsinghua University (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
August 27, 2021

Certified by . . . .
Leslie Pack Kaelbling
Panasonic Professor of Computer Science and Engineering
Thesis Supervisor

Certified by . . . .
Joshua B. Tenenbaum
Professor of Computational Cognitive Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Learning, Reasoning, and Planning with Relational and Temporal Neural Networks

by

Jiayuan Mao

Submitted to the Department of Electrical Engineering and Computer Science on August 27, 2021, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

Abstract

Every day, people interpret events and actions in terms of concepts, defined over evolving relations among agents and objects and their goals. We learn these concepts from a limited amount of data, generalizing directly over different numbers and arrangements of agents and objects, and detailed timings of trajectories. We also effectively recompose these concepts to describe unseen behaviors from other agents, and leverage the causal relationships among actions to make plans for ourselves.

This thesis gives an overview of a neuro-symbolic framework for learning, reasoning, and planning with relational and temporal neural networks. The key idea is to exploit a structural bias in neural network learning that enables us to describe complex relational-temporal events and actions. These structures form a minimal amount of prior knowledge but are generic and crucial: scenes are composed of objects; events are temporally related; actions have preconditions and goals. Our systems learn from trajectories with rich temporal and relational patterns and labels for events and actions. We demonstrate that they can generalize from small amounts of data to scenarios containing more objects than were present during training and to temporal warpings of input sequences, and exploits the goal-centric representation of actions to make plans for novel goals.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Panasonic Professor of Computer Science and Engineering

Thesis Supervisor: Joshua B. Tenenbaum
Title: Professor of Computational Cognitive Science

// Acknowledgments

I want to express my deepest appreciation to my advisors, Professor Leslie Kaelbling and Professor Joshua Tenenbaum. Leslie and Josh have always been inspiring and supportive. I have learned so much from their invaluable insights and ideas on both meta-level—how to find interesting research questions and how to formulate them, and the concrete level—how different problems relate with each other and what are principled ways to approach them. I have also learned countless lessons about life by working and interacting with them: always being enthusiastic about research, staying open-minded and critical to ourselves, and trying hard to balance work and life.

I would also like to extend my deepest gratitude to Professor Jiajun Wu for being my mentor and friend since my undergraduate years. I was very fortunate to learn many lessons from him on almost every aspect of research: formulation, implementation, collaboration, and presentation. I am also extremely grateful to Professor Tomás Lozano-Pérez for sharing his profound insights on robotics, and, more broadly, AI, as well as his awesome jokes that make those stressful research meetings so entertaining.

I am also very grateful to have guidance and support from my undergraduate advisors and mentors: Yuning Jiang and Dr. Zhimin Cao for introducing me into the world of machine learning and computer vision, Professor Joseph Lim for advising me at USC and motivating me to think hard about challenges in artificial intelligence, Dr. Denny Zhou, Dr. Lihong Li, Dr. Chong Wang for mentoring me at Google AI and introducing me to the problem of compositional generalization, Professor Zhiyuan Liu for supervising my undergraduate thesis and introducing me to the world of natural language understanding. They have played a decisive role in shaping my research through their ideas, advice, and encouragement.

All work in this thesis was made possible by my awesome collaborators Honghua Dong and Zhezheng Luo. I also want to thank my group members at the MIT Learning and Intelligent Systems (LIS) group, the MIT Computational Cognitive Science (CoCoSci) group: Yilun Du, Xiaolin Fang, Zhutian Yang, Catherine Wong, Dr. Chuang Gan, Dr. Caelan Garrett, Dr. Zi Wang, Ferran Alet, Rachel Holladay, and Sahit Chintalapudi, Tan Zhi-Xuan, Yoni Friedman, and my colleagues Xiuming Zhang, Yikai Li, Ruocheng Wang, Sumith Kulal, Zhenfang Chen, Jiawei Yang, Chi Han, Sidi Lu, and Hao Wu. I am also grateful to Freda Shi, who has been my long-term friend and research collaborator, for inspiring me, discussing ideas with me, and teaching me natural language processing.

I would also like to extend my gratitude to my dear friends for supporting me and

encouraging me throughout my life.

I am deeply indebted to Yanzi Han, for her iridescent love, support, and encouragement bestowed upon me, for standing behind me during my toughest times, and for having been there with me to make plans for the future.

I received the MIT Presidential Fellowship during my first year. I gratefully acknowledge support from the Center for Brains, Minds and Machines (NSF STC award CCF1231216), from ONR MURI N00014-16-1-2007; from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from the Honda Research Institute, from MIT-IBM Watson Lab; and SUTD Temasek Laboratories.

Finally, many thanks to my parents, for their unconditional love.

Contents

1 Introduction **15**

2 Preliminary: Neural Logic Machines **17**
2.1 Introduction 17
2.2 Neural Logic Machines (NLM) 19
2.2.1 Logic Predicates as Tensors 20
2.2.2 Logic Rules as Neural Operators 20
2.2.3 Neural Logic Machines 23
2.2.4 Expressiveness and Computational Complexity 25

3 Learning to Describe Events with Temporal and Object Quantification Networks **27**
3.1 Introduction 27
3.2 TOQ-Nets 30
3.2.1 Temporal-Relational Feature Representation 31
3.2.2 Relational Reasoning Layers 33
3.2.3 Temporal Reasoning Layers 34
3.3 Experiments 36
3.3.1 Baseline Approaches 37
3.3.2 Trajectory-Based Soccer Event Detection 37
3.3.3 Manipulation Concept Learning from 6D Poses 42
3.3.4 Extension to Real-World Datasets 45
3.4 Related Work 47
3.5 Conclusion and Discussion 48

4 Learning to Plan with Skills from Rational Demonstrations **49**
4.1 Introduction 49
4.2 Related Work 51
4.3 Planning and Learning of RatSkills 52

4.3.1 Problem Formulation . . . 52
4.3.2 Task-Augmented Transition Models . . . 54
4.3.3 Planning and Inverse Planning with RatSkills . . . 55
4.3.4 Learning RatSkills . . . 58
4.4 Experiments . . . 59
4.4.1 Setup . . . 59
4.4.2 Environments . . . 60
4.4.3 Results . . . 61
4.5 Conclusion . . . 65

5 Conclusion **67**

A Implementation Details for TOQ-Nets **69**
A.1 Implementation Details . . . 69
A.1.1 TOQ-Nets . . . 69
A.1.2 STGCN . . . 70
A.1.3 STGCN-MAX . . . 70
A.1.4 STGCN-2D . . . 71
A.1.5 STGCN-LSTM . . . 71
A.1.6 Space-Time Region Graph Networks . . . 72
A.1.7 Non-Local Neural Networks . . . 72

B Details and Results for RatSkills **77**
B.1 Dataset . . . 77
B.1.1 Crafting World . . . 77
B.1.2 Playroom . . . 80
B.2 Implementation Details . . . 82
B.2.1 LSTM . . . 82
B.2.2 Inverse Reinforcement Learning (IRL) . . . 82
B.2.3 Behavior Cloning (BC) . . . 84
B.2.4 Behavior Cloning with FSM (BC-FSM) . . . 84
B.2.5 FSM-AStar . . . 85

List of Figures

2-1 (Top) A graphical illustration of the blocks world. Given an initial and a target worlds, the agent is required to move blocks to transform the initial configuration to the target one. (Down) A set of sentences used throughout the chapter to define the blocks world. 19

2-2 An illustration of Neural Logic Machines (NLM). During forward propagation, NLM takes object properties and relations as input, performs sequential neural network computations, and outputs properties or relations of the objects. Implementation details can be found in Section 2.2.3. 21

2-3 An illustration of the computational block inside NLM for binary predicates at layer i. $C_i^{(j)}$ denotes the number of output predicates of group j at layer i. $[\cdot]$ denotes the shape of the tensor. 24

3-1 (a) An input sequence composed of relational states: each column represents the state of an entity that changes over time. A logic formula describes a complex concept or feature that is true of this temporal sequence using object and temporal quantification. The sequence is segmented into three stages: throughout the first stage, ■ holds for at least one entity, until the second stage, in which each entity is always either ■ or ■, until the third stage, in which ■ eventually becomes true for at least one of the entities. (b) Such events can be described using first-order linear temporal logic expressions. 28

3-2 A TOQ-Net contains three modules: (i) an input feature extractor, (ii) relational reasoning layers, and (iii) temporal reasoning layers. To illustrate the model's representational power, we show that logical forms of increasing complexity can be realized by stacking multiple layers. . 30

3-3 Illustration of (i) relational reasoning layers and (ii) temporal reasoning layers. We provide two illustrative running traces. (i) The first relational reasoning layer takes unary predicates q_1 and q_2 as input and its output Q_1 is able to represent $q_1 \wedge q_2$. The $\max(Q_1, dim = 0)$ in layer 2 can represent $\exists x.\ q_1(x, t) \wedge q_2(x, t)$. (ii) Assume P_K encodes the occurance of events e_1 and e_2 at each time step. The first temporal reasoning layer can realize *always* e_2 with a temporal pooling from time step 3 to time step T. In the second temporal reasoning layer, the temporal pooling summarizes that e_1 holds true from time step 1 to 2. Thus, the NN should be able to realize e_1 *until* (*always* e_2). 32

3-4 Generalization to soccer environments with a different court size and agent speeds. The standard errors are computed based on three random seeds. 40

3-5 Comparing # of reasoning layers. Accuracy is tested on 6v6 9-way classification. When # of relational reasoning layers vary (blue), temporal reasoning layers are fixed at 4. When # of temporal reasoning layers vary (orange), relational reasoning layers are fixed at 3. When we have 0 temporal reasoning layers, we predict the sequence label based on the feature of the frame of interest. The purple line shows the performance on temporally warped trajectories when we have the # of relational reasoning layers fixed and vary the # of temporal reasoning layers. Relational layers are required for the prediction task (# of relational reasoning layer must be greater than 0), because there is no nullary feature input to the network: we need at least one relational reasoning layer to gather information across the entities. Overall, adding reasoning layers improves results, and 3 relational layers + 4 temporal layers is a good balance between computation and performance. 43

3-6 Relevant features in temporal layers. Feature dependencies are computed by gradient. These dependencies and thresholds are learned end-to-end from data. Insets detail features for events in the first and the second stage of the *high pass* and *short pass*. 44

3-7 Comparing different models with different time stretching factors on the RLBench dataset. 46

4-1 Interpreting a demonstration and its description in terms of RatSkills: (a) Each RatSkill consists of two conditions I_o and G_o. (b) The system infers a transition to the next skill if both the G condition of the current skill and the I condition of the next skill are satisfied. Such transition rules can be used to interpret demonstrations and to plan for tasks that require multiple skills to achieve. 50
4-2 Illustrative example of how finite state machines (FSM) are constructed for tasks. 54
4-3 An example of the value function $J_t(s, v, a)$ for task-augmented states on a simple FSM. $\max_{a \in \mathcal{A}} J_t(s, v, a)$ are plotted at each location at each FSM node. Deeper color indicates larger cost. Dotted lines illustrate one *rational* trajectory for each skill; red boxes indicate goals. 57
4-4 An illustration of the Playroom environment and a trajectory for the task: *turn on the music then play the ball then turn off the music*. . . 61
4-5 Planning success rate on 12 novel tasks in the Crafting World environments, sorted by the number of skills in the task description. We use different colors to represent environments with different obstacles. Green: no obstacles. Yellow: there are doors to be opened by specific keys. Blue: the agent must craft a boat to go across rivers. Orange: the environment has both doors and rivers. 63
4-6 RatSkills can be integrated into a simple forward-search algorithm to improve the success rate w.r.t. the same number of expanded nodes in the search tree, because the learned skill models suggest meaningful subgoal states. We do evaluation on 3 planning tasks in the Crafting World environment. We use 100 random initial states for each task. . 64

B-1 A running example of the FSM-A^* algorithm for the task "*(mine wood* ***or*** *mine coal)* ***then*** *mine gold*." For simplicity, we only show a subset of states visited on each FSM node. The blue arrows indicate transitions by primitive actions (in this example, each primitive action takes a cost of 0.1). The yellow arrows are transitions on the FSM, which can only be performed when $G_v(\cdot)$ and $I_{v'}(\cdot)$ evaluates to True (in practice, the reward is computed as $-(\log \Pr(G_v(\cdot) + \log \Pr(I_{v'}(\cdot))))$. At the super-terminal node v_T, the state with minimum cost will be selected and we will back-trace the entire state-action sequence. 86

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

3.1 A running example of different temporal quantification formulas that TOQ-Nets can realize. For clarity, we use T and F for True/False. In the actual computation, they are "soft" Boolean values ranges in $[0, 1]$. **G** means ***always***; **F** means ***Eventually***; **U** means ***until***. 35

3.2 Results on the soccer event dataset. Different columns correspond to different action sets (the regular, few-shot, and full action sets). The performance is measured by per-action (macro) accuracy, averaged over nine few-shot splits. The ± values indicate standard errors. TOQ-Net significantly outperforms all baseline methods on the few-shot action set. 37

3.3 Full results on generalization to scenarios with all combinations of #agents and temporally warping on the soccer event dataset. The standard error of all values are smaller than 2.5%, computed based on three random seeds. 39

3.4 Results on the soccer event dataset for baselines with different capacities, where model_S and model_T denote the small and the tiny variant for each model. The performance is measured by per-action (macro) accuracy, averaged over nine few-shot splits. The ± values indicate standard errors. For each baseline we showed the best performance over three levels of capacities. In fact, performances of most of the models are not affected much by the capacity. We highlight the best-performing variants of all baselines, and use them in all comparisons in the main text. 41

3.5 Few-shot learning on the RLBench dataset, measured by per-action (macro) accuracy and averaged of four 1-shot splits and four random seeds per split. The ± values indicate standard errors. On the right we shows the sampled performance of different models on each individual 1-shot split. 45

4.1 Results of the planning task for the Crafting World environment and the Playroom environment. All models are trained on two data splits: *primitive* and *compositional*. They are evaluated on the *compositional* and the *novel* split, which contains unseen task descriptions. 62
4.2 Results of the inverse planning task for the Crafting World environment and the Playroom environment. The goal is to predict the most probable intended task of the agent by observing their state-action sequences. All models are trained on the union of the *primitive* and *compositional* data splits and evaluated on the *compositional* and the *novel* split. . . 64

A.1 Ablation study of the STGCN-2D model on the soccer event dataset, evaluated by its few-shot learning performance. 71
A.2 Ablation study of the STGCN-2D model on the soccer event dataset, evaluated by its generalization to different number of players and time-warped trajectories. 71
A.3 Ablation study of different kernel sizes for the STGCN-LSTM model on the RLBench dataset, measured by per-action (macro) accuracy and averaged of four 1-shot splits and four random seeds per split. The $\pm$ values indicate standard errors. 72
A.4 The STGCN architecture used in the paper. 74
A.5 The Space-Time Region Graph architecture used in the paper. 75
A.6 The Non-local Neural Networks architecture used in the paper. 75

B.1 Task descriptions in the *primitive*, *compositional* and *novel* sets for the Crafting World. 80
B.2 Task descriptions in the *primitive*, *compositional* and *novel* sets for the Playroom. 81

Chapter 1

Introduction

Humans understand events and actions in terms of concepts, defined over temporally evolving relations among agents and objects and their goals. For example, in a soccer game, the action *passing* the ball is recognized as a series of actions that changes the ball ownership, with the goal of letting a teammate take control of the ball. Such temporal and goal-centric representation of actions allows us to recognize actions that happen in the field, interpret the strategy of soccer players, and furthermore, enables ourselves to make plans for scoring in a soccer game.

We have seen a significant advancement of deep neural networks in various artificial intelligence (AI) tasks in recent years, especially in the field of recognizing actions and making decisions in interactive domains. However, compared with humans, they still need a tremendous amount of data to recognize an object's properties, or to acquire a new skill. Moreover, it is usually unclear how these learned concepts can generalize compositionally to novel situations (for example, from a 3v3 soccer game to an 11v11 game), or how these learned skills can be recombined to achieve novel goals.

To address these challenges, this thesis focuses on developing a neuro-symbolic framework for describing the relational, temporal, and goal-centric structures of events and actions. It marries the ability of neural networks to learn latent representations from data, with the expressiveness of logic languages, specifically first-order linear temporal logic (FO-LTL), in describing temporal structures of events and actions. The key idea is to exploit a structural bias in neural networks that enables us to describe complex relational-temporal events. These structures form a minimal amount of prior knowledge but are generic and crucial: scenes are composed of objects; events are temporally related; actions have preconditions and goals.

The presented framework exhibits two important desiderata of a machine learning system: data efficiency and compositional generalization. It learns new concepts from a limited amount of data, and generalizes compositionally to scenarios with a

different number of agents and objects, and detailed timings of trajectories. With an abstract temporal language, humans can recompose concepts learned by the machine in novel ways to form novel concepts or tasks. We believe this framework will serve as a foundation for more capable robots that can continually learn new knowledge from their experiences and apply the knowledge in their reasoning and planning in the physical world.

The thesis is composed of three chapters. In Chapter 2, we revisit prior work on modeling object-centric relational structures with neural networks. It will serve as a foundational component in our modeling of relational and temporal structures for events and actions. In Chapter 3, we extend these ideas to modeling events as a temporal composition of relational states and changes. In Chapter 4, we discuss a goal-centric representation for actions in interactive environments, and further combine learning with planning algorithms to solve unseen tasks that require multiple steps to accomplish.

Chapter 2

Preliminary: Neural Logic Machines

This chapter introduces the Neural Logic Machine (NLM), a neuro-symbolic architecture for learning rules from relational data. NLMs exploit the power of both neural networks—as function approximators, and logic programming—as a symbolic processor for objects with properties, relations, logic connectives, and quantifiers. After being trained on small-scale tasks (such as sorting short arrays), NLMs can recover lifted rules, and generalize to large-scale tasks (such as sorting longer arrays). NLM will serve as a foundational component in our modeling of relational and temporal structures for evens and actions in later chapters.

The material of this chapter has been previously published in Dong *et al.* [2019]. Honghua Dong, in particular, contributed significantly to the materials presented in this chapter.

2.1 Introduction

Deep learning has achieved great success in various applications such as speech recognition [Hinton *et al.*, 2012], image classification [Krizhevsky *et al.*, 2012; He *et al.*, 2016], machine translation [Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015; Wu *et al.*, 2016; Vaswani *et al.*, 2017], and game playing [Mnih *et al.*, 2015; Silver *et al.*, 2017]. Starting from Fodor and Pylyshyn [1988], however, there has been a debate over the problem of systematicity (such as understanding recursive systems) in connectionist models [Fodor and McLaughlin, 1990; Hadley, 1994; Jansen and Watter, 2012].

Logic systems can naturally process symbolic rules in language understanding and reasoning. Inductive logic programming (ILP) [Muggleton, 1991, 1996; Friedman *et*

al., 1999] has been developed for learning logic rules from examples. Roughly speaking, given a collection of positive and negative examples, ILP systems learn a set of rules (with uncertainty) that entails all of the positive examples but none of the negative examples. However, this is a fundamentally challenging task as the hypothesis space of possible rules is exponentially large [Dantsin *et al.*, 2001; Lin *et al.*, 2014; Evans and Grefenstette, 2018].

To make the discussion concrete, let us consider the classic blocks-world problem [Nilsson, 1982; Gupta and Nau, 1992]. As shown in Fig. 2-1, we are given a set of blocks on the ground. We can move a block x and place it on the top of another block y or the ground, as long as x is *moveable* and y is *placeable*. We call this operation `Move`(x, y). A block is said to be *moveable* or *placeable* if there are no other blocks on it. The ground is always *placeable*, implying that we can place all blocks on the ground. Given an initial configuration of blocks-world, our goal is to transform it into a target configuration by taking a sequence of `Move` operations.

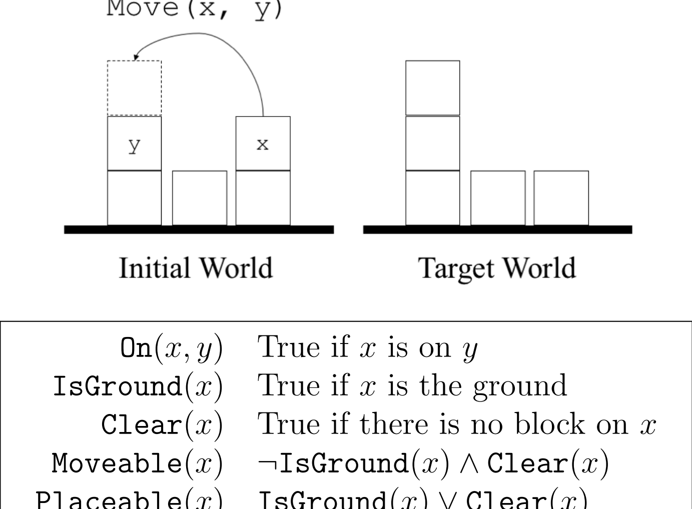
Although the blocks-world problem may appear simple at first glance, four major challenges exist in building a learning system to automatically accomplish this task:

1. The learning system should recover a set of lifted rules (i.e., rules that apply to objects generically instead of being tied to specific ones) and generalize to blocks-worlds which contain more blocks than those encountered during training. To get intuition about this, we refer the readers who are not familiar with the blocks-world domain to the task of learning to sort arrays [e.g., Vinyals *et al.*, 2015], where recurrent neural networks fail to generalize to arrays which are even just slightly longer than those for training.

2. The learning system should deal with high-order relational data and quantifiers, which goes beyond the scope of typical graph-structured neural networks [Kipf and Welling, 2017], which only consider binary relationships between objects. For example, to apply the transitivity rule of a relation r , i.e. $r(a, c) \leftarrow \exists b\; r(a, b) \wedge r(b, c)$, we need to jointly inspect three objects (a, b, c).

3. The learning system should scale up w.r.t. the complexity of the rules. Existing logic-driven approaches such as traditional ILP methods suffer an exponential computational complexity w.r.t. the number of logic rules to be learned [Dantsin *et al.*, 2001; Lin *et al.*, 2014; Evans and Grefenstette, 2018].

4. The learning system should recover rules based on a minimal set of learning priors. In contrast, traditional ILP methods usually require hand-coded and task-specific rule templates to restrict the size of searching spaces [Evans and Grefenstette, 2018].



On(x, y)	True if x is on y
IsGround(x)	True if x is the ground
Clear(x)	True if there is no block on x
Moveable(x)	$\neg$IsGround$(x) \wedge$ Clear(x)
Placeable(x)	IsGround$(x) \vee$ Clear(x)

Figure 2-1: (Top) A graphical illustration of the blocks world. Given an initial and a target worlds, the agent is required to move blocks to transform the initial configuration to the target one. (Down) A set of sentences used throughout the chapter to define the blocks world.

In this chapter, we present Neural Logic Machines (NLMs) to address the aforementioned challenges. In a nutshell, NLMs are neuro-symbolic architectures that can realize first-order logic (FOL) rules in a finite domain. The key intuition behind NLMs is that logic operations such as logical ANDs and ORs can be realized with feed-forward neural networks, and the wiring among neural modules can realize the logic quantifiers, such as universal and existential quantifiers.

2.2 Neural Logic Machines (NLM)

The NLM is a neural realization of logic machines over finite domains and under the Closed-World Assumption*. Given a set of base predicates, grounded on a set of objects (the *premises*), NLMs sequentially apply first-order rules to draw *conclusions*, such as a property about an object. For example, in the blocks world, based on premises IsGround(u) and Clear(u) of object u, NLMs can infer whether u is moveable.

Internally, NLMs use tensors to represent logic predicates. This is done by grounding the predicate as True or False over a fixed set of objects. Based on the tensor representation, rules are implemented as neural operators that can be applied over the premise tensors and generate conclusion tensors. Such neural operators can handle relational data with various orders (i.e., operating on predicates with different arities).

*https://en.wikipedia.org/wiki/Closed-world_assumption

2.2.1 Logic Predicates as Tensors

We use tensors to represent the grounding of logic predicates. Suppose we have a universe of objects $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$. A predicate $p(x_1, x_2, \ldots, x_r)$, of *arity* r, can be grounded on the object set $\mathcal{U}$ (informally, we call it $\mathcal{U}$*-grounding*), resulting in a tensor $p^{\mathcal{U}}$ of shape $[m^{\underline{r}}] \triangleq [m, m-1, m-2, \ldots, m-r+1]$, where the value of each entry $p^{\mathcal{U}}(u_{i_1}, u_{i_2}, \ldots, u_{i_r})$ of the tensor represents whether p is `True` under the grounding that $x_1 = u_{i_1}, x_2 = u_{i_2}, \cdots, x_r = u_{i_r}$. Here, we restrict that the grounded objects of all x_i's are mutually exclusive, i.e., $i_j \neq i_k$ for all pairs of indices j and k. This restriction does not limit the generality of the representation, as the "missing" entries can be represented by the $\mathcal{U}$-grounding of other predicates with a smaller arity. For example, for a binary predicate p, the grounded values of the $p^{\mathcal{U}}(x, x)$ can be represented by the $\mathcal{U}$-grounding of a unary predicate $p'(x) \triangleq p(x, x)$.

We extend this representation to a collection of predicates of the same arity. Let $C^{(r)}$ be the number of predicates of arity r. We stack the $\mathcal{U}$*-grounding* tensors of all predicates as a tensor of shape $\left[m^{\underline{r}}, C^{(r)}\right] \triangleq \left[m, m-1, m-2, \ldots, m-r+1, C^{(r)}\right]$, where the last dimension corresponds to the predicates. Intuitively, a group of $C^{(1)}$ unary predicates grounded on m objects can be represented by a tensor of shape $\left[m, C^{(1)}\right]$, describing a group of "properties of objects", while a $\left[m, m-1, C^{(2)}\right]$-shaped tensor for $C^{(2)}$ binary predicates describes a group of "pairwise relations between objects". In practice, we set a maximum arity B for the predicates of interest, called the *breadth* of the NLM.

Each entry in $\mathcal{U}$*-grounding* tensors takes value from $[0, 1]$, which can be interpreted as a score indicating whether the predicate is `True`, grounded on a specific set of objects. All premises, conclusions, and intermediate results in NLMs are represented by such tensors. Such "softening" enables us to directly use gradient descent to optimize the weights. As a side note, we impose the restriction that all arguments in the predicates can only be variables or objects (i.e., constants) but not function symbols, which follows the setting of *Datalog* [Maier and Warren, 1988].

2.2.2 Logic Rules as Neural Operators

Our goal is to build a neural architecture to learn rules that are both lifted and able to handle relational data with multiple arities. We present different modules of our neural operators by making analogies to a set of essential *meta-rules* in symbolic logic systems. Specifically, we discuss our how our neural architecture can realize (1) *boolean logic rules*, as lifted rules containing boolean operations (`AND`, `OR`, `NOT`) over a set of predicates; and (2) *quantifications*, which bridge predicates with different arities

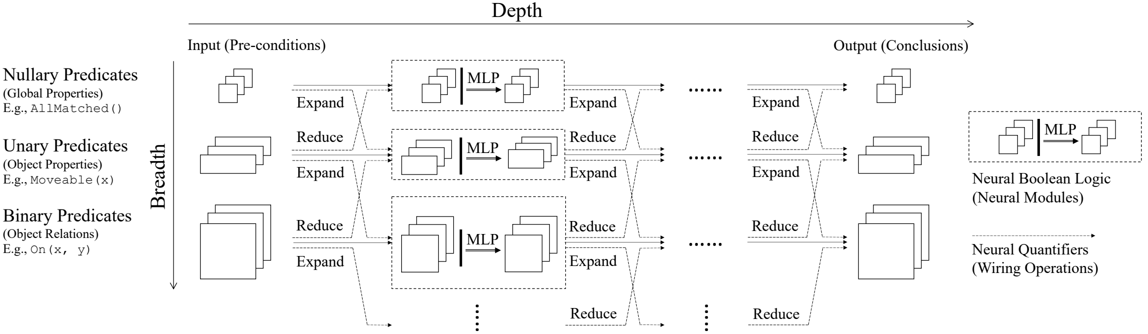


Figure 2-2: An illustration of Neural Logic Machines (NLM). During forward propagation, NLM takes object properties and relations as input, performs sequential neural network computations, and outputs properties or relations of the objects. Implementation details can be found in Section 2.2.3.

by logic quantifiers (∀ and ∃).

Next, we combine these neural units to compose NLMs. Figure 2-2 illustrates the overall multi-layer, multi-group architecture of an NLM. An NLM has layers of *depth* D (horizontally), and each layer has $B+1$ computation units (vertically). These units operate on the tensor representations of predicates whose arities range from $[0, B]$, respectively. NLMs take input tensors of predicates (premises), perform layer-by-layer computations, and output tensors as conclusions.

As the number of layers increases, higher levels of abstraction can be formed. For example, the output of the first layer may represent $\texttt{Clear}(x)$, while a deeper layer may output a more complicated predicate like $\texttt{Moveable}(x)$. Thus, forward propagation in NLMs can be interpreted as a sequence of rule applications. We further show that NLMs can efficiently realize a partial set of Horn clauses.

We start from the *neural boolean logic rules* and the *neural quantifiers*.

Boolean logic . We use the following symbolic meta-rule for boolean logic:

$$\hat{p}(x_1, x_2, \cdots, x_r) \leftarrow \texttt{expression}(x_1, x_2, \cdots, x_r), \tag{2.1}$$

where `expression` can be any *boolean* expressions consisting of predicates over *all* variables $(x_1, \ldots, x_r)$ and $\hat{p}(\cdot)$ is the derived predicate. For example, the rule $\texttt{Moveable}(x) \leftarrow \neg\texttt{IsGround}(x) \wedge \texttt{Clear}(x)$ can be instantiated from this meta-rule.

Denote $\mathcal{P} = \{p_1, \ldots, p_k\}$ as the set of $|\mathcal{P}|$ predicates appeared in `expression`. By definition, all p_i's have the same arity r and can be stacked as a tensor of shape $[m^{\underline{r}}, |\mathcal{P}|]$. In Eq. 2.1, for a specific grounding of the derived predicate $\hat{p}(x_1 \cdots x_r)$, it

is conditioned $r! \times |\mathcal{R}|$ grounding values with the same subset of objects, of *arbitrary permutation* as the arguments to all input predicates $\mathcal{P}$. For example, consider a specific ternary predicate $\hat{p}(x_1, x_2, x_3)$. For three different objects $a, b, c \in \mathcal{U}$, the grounding $\hat{p}(a, b, c)$ is conditioned on $p_j(a, b, c), p_j(a, c, b), p_j(b, a, c), p_j(b, c, a), p_j(c, a, b), p_j(c, b, a)$ (all permutations of the parameters) for all j (all input predicates).

Our neural implementation of boolean logic rules is a lifted neural module that uniformly applies to any grounding entries $(x_1 \cdots x_r)$ in the output tensor $\hat{p}^{\mathcal{U}}$. It has a Permute$(\cdot)$ operation transforming the tensor representation of $\mathcal{P}$, followed by a multi-layer perceptron (MLP). Given the tensor representation of $\mathcal{P}$, for each $p_i^{\mathcal{U}}(x_1, x_2, \ldots, x_r)$, the Permute$(\cdot)$ operation creates $r!$ new tensors as $p_{i,1}^{\mathcal{U}}, \ldots, p_{i,r!}^{\mathcal{U}}$ by permuting all axes that index objects, with all possible permutations. We stack all to form a $[m^{\underline{r}}, r! \times |\mathcal{P}|]$-shaped tensor. An MLP uniformly applies to all $m^{\underline{r}}$ object indices:

$$\hat{p}(u_{i_1}, \cdots, u_{i_r}) = \sigma \left(\text{MLP} \left(p_{1,1}(u_{i_1}, \ldots, u_{i_r}), \cdots, p_{k,r!}(u_{i_1}, \ldots, u_{i_r}) \right); \theta \right), \tag{2.2}$$

where σ is the sigmoid nonlinearity, θ is the trainable network parameters. For all sets of mutually exclusive indexes $i_1, \ldots, i_r \in \{1, 2, \ldots, m\}$, the same MLP is applied. Thus, the size of θ is independent of the number of objects m. This property is analogous to the implicit unification property of Horn clauses: the rule $\hat{p}(x) \leftarrow p_1(x) \wedge p_2(x)$ implicitly means, $\forall x \; \hat{p}(x) \leftarrow p_1(x) \wedge p_2(x)$.

Quantification . We introduce two types of meta-rules for quantification, namely *expansion* and *reduction*. Let p be a predicate, and we have

(Expansion) $$\forall x_{r+1} \; q(x_1, x_2, \cdots, x_r, x_{r+1}) \leftarrow p(x_1, x_2, \cdots, x_r), \tag{2.3}$$

where $x_{r+1} \notin \{x_i\}_{i=1}^{r}$. The expansion operation constructs a new predicate q from p, by introducing a new variable x_{r+1}. For example, consider the following rule

$$\texttt{ValidMove}(x, y) \leftarrow \texttt{Moveable}(x) \wedge \texttt{Placeable}(y).$$

This rule does not fit the meta-rule in Eq. 2.1 as some predicates on the RHS only take a *subset* of variables as inputs. However, it can be described by using the expansion and the boolean logic meta-rules jointly.

1. $\forall z \; \texttt{MoveableX}(x, z) \leftarrow \texttt{Moveable}(x)$;	(from Eq. 2.3)
2. $\forall z \; \texttt{PlaceableY}(y, z) \leftarrow \texttt{Placeable}(y)$;	(from Eq. 2.3)
3. $\texttt{ValidMove}(x, y) \leftarrow \texttt{MoveableX}(x, y) \wedge \texttt{PlaceableY}(y, x)$.	(from Eq. 2.1)

The *expansion meta-rule* (Eq. 2.3) for a set of C r-ary predicates, represented by a $[m^{\underline{r}}, C]$-shaped tensor, introduces a new and distinct variable x_{r+1}. Our neural implementation Expand($\cdot$) repeats each predicate (their tensor representation) for $(m-r)$ times, and stacks in a new dimension. Thus the output shape is $[m^{\underline{r+1}}, C]$.

The other meta-rule is for reduction:

$$\textbf{(Reduction)} \qquad q(x_1, x_2, \cdots, x_r) \leftarrow \forall x_{r+1}\ p(x_1, x_2, \cdots, x_r, x_{r+1})\,, \tag{2.4}$$

where the $\forall$ quantifier can also be replaced by $\exists$. The reduction operation reduces a variable in a predicate via the quantifier. As an example, the rule to deduce the moveability of objects,

$$\texttt{Moveable}(x) \leftarrow \neg\texttt{IsGround}(x) \wedge \neg\big(\exists y\ \texttt{On}(y, x)\big)\,,$$

can be expressed using meta-rules as follows:

1. $\texttt{Clear}(x) \leftarrow \forall y\ \neg\texttt{On}(y, x)$;	(from Eq. 2.4)
2. $\texttt{Moveable}(x) \leftarrow \neg\texttt{IsGround}(x) \wedge \texttt{Clear}(x)$.	(from Eq. 2.1)

The *reduction meta-rule* (Eq. 2.4) for a set of C $(r+1)$-ary predicates, represented by a $[m^{\underline{r+1}}, C]$-shaped tensor, eliminates the variable x_{r+1} via quantifiers. For $\exists$ (or $\forall$), our neural implementation Reduce($\cdot$) takes the maximum (or minimum) element along the dimension of x_{r+1}, and stacks the two resulting tensors. Therefore, the output shape becomes $[m^{\underline{r}}, 2C]$.

2.2.3 Neural Logic Machines

NLMs realize symbolic logic rules in a multi-layer multi-group architecture, illustrated in Fig. 2-2. An NLM has D layers, and each layer has $B+1$ computation units as groups. Between layers, we use *intra-group computation* (Eq. 2.1). The predicates at each layer are grouped by their arities, and inside each group, we use *inter-group computation* (Eq. 2.3 and 2.4).

We define $\mathcal{O}_i = \left\{O_i^{(0)}, O_i^{(1)}, \cdots, O_i^{(B)}\right\}$ as the outputs of layer i, where $O_i^{(r)}$ is the output corresponding to the r-ary unit at layer i. For convenience, we denote $\mathcal{O}_0 = \left\{O_0^{(0)}, O_0^{(1)}, \cdots, O_0^{(B)}\right\}$ as the $\mathcal{U}$-grounding tensors for NLM's base predicates (the *premises*), and $\mathcal{O}_D$ at the last layer as the *conclusions*. The overall computation is performed layer-by-layer, from layer 1 to layer D. All computation units at layer i work simultaneously, taking $\mathcal{O}_{i-1}$ as inputs and generating $\mathcal{O}_i$.

Let us consider a specific group r at layer i, and we show how to calculate $O_i^{(r)}$.

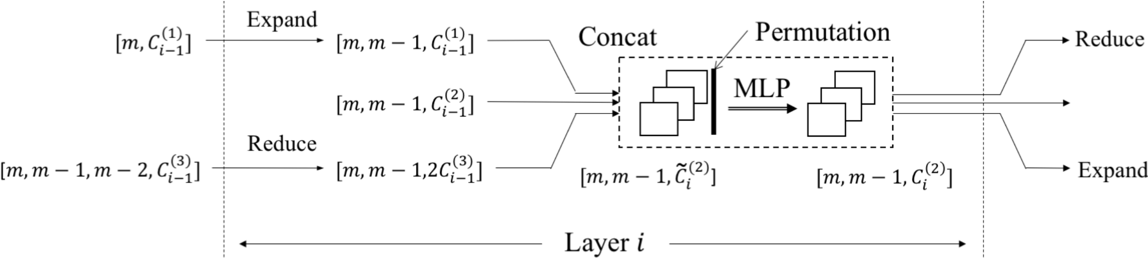


Figure 2-3: An illustration of the computational block inside NLM for binary predicates at layer i. $C_i^{(j)}$ denotes the number of output predicates of group j at layer i. $[\cdot]$ denotes the shape of the tensor.

Inter-group computation. As shown in Figures 2-2 and 2-3, we connect tensors from the previous layer $i-1$ in vertically neighboring groups (i.e. $r-1$, r and $r+1$), and aligns their shapes by *expansion* (Eq. 2.3) or *reduction* (Eq. 2.4) to form an intermediate tensor $I_i^{(r)}$:

$$I_i^{(r)} = \text{Concat}\left(\text{Expand}\left(O_{i-1}^{(r-1)}\right), O_{i-1}^{(r)}, \text{Reduce}\left(O_{i-1}^{(r+1)}\right)\right). \tag{2.5}$$

Nonexistent terms are ignored (e.g. when $r+1 > B$ or $r-1 < 0$). Note that from the previous layer, $O_{i-1}^{(r-1)}, O_{i-1}^{(r)}, O_{i-1}^{(r+1)}$ have shapes $[m^{\underline{r-1}}, C_{i-1}^{(r-1)}]$, $[m^{\underline{r}}, C_{i-1}^{(r)}]$, $[m^{\underline{r+1}}, C_{i-1}^{(r+1)}]$, respectively. After the concatenation, the resulting tensor $I_i^{(r)}$ is of shape $[m^{\underline{r}}, \widetilde{C}_i^{(r)}]$, where the number of new predicates is $\widetilde{C}_i^{(r)} \triangleq C_{i-1}^{(r-1)} + C_{i-1}^{(r)} + 2C_{i-1}^{(r+1)}$, and the 2 comes from the two quantifiers ($\forall$ and $\exists$).

The inter-group computation essentially aligns predicates of neighboring arities. Relational representations of different orders get combined together through the neural quantification.

Intra-group computation. The intra-group computation is implemented as the neural boolean logic in Eq. 2.1. It take the intermediate tensor $I_i^{(r)}$ as input, permutes and generates the output tensor $O_i^{(r)}$:

$$O_i^{(r)} = \sigma\left(\text{MLP}\left(\text{Permute}\left(I_i^{(r)}\right); \theta_i^{(r)}\right)\right), \tag{2.6}$$

where σ is the sigmoid nonlinearity and $\theta_i^{(r)}$ denotes trainable parameters. We apply Permute function to $\widetilde{C}_i^{(r)}$ tensors in $I_i^{(r)}$ individually, and get $r!\widetilde{C}_i^{(r)}$ tensors. We set the number of output neurons to be $C_i^{(r)}$, thus the shape of output tensor $O_i^{(r)}$ is $[m^{\underline{r}}, C_i^{(r)}]$.

Example. For concreteness, in Fig. 2-3, consider group 2 (*binary* predicates) at layer i. The module begins with the inter-group computation. It first collects the output of vertically consecutive groups (unary, binary and ternary) from the previous layer $i-1$, where their shapes are shown in the figure. Then it uses expansion/reduction to compose the intermediate tensor $I_i^{(2)}$ containing $\widetilde{C}_i^{(2)} \triangleq C_{i-1}^{(1)} + C_{i-1}^{(2)} + 2C_{i-1}^{(3)}$ predicates. For each object pair (x, y), the output $\mathcal{U}$-grounding tensor of predicates is computed by intra-group computation $O_i^{(2)}(x, y) = \text{MLP}(\text{Concat}(I_i^{(2)}(x, y), I_i^{(2)}(y, x)); \theta_i^{(2)})$, and the output shape is $[m, m-1, C_i^{(2)}]$. The $\text{Concat}(\cdot, \cdot)$ corresponds to the `Permute` operation, while the MLP is shared among all pairs of objects (x, y).

Remark. It can be verified that NLMs can realize the forward chaining of a partial set of Horn clauses. In NLMs, we consider only finite cases. Thus, there should not exist cyclic references of predicates among rules. The extension to support cyclic references is left as a future work. Thus, given the training dataset containing pairs of (*premises*, *conclusions*), NLMs can induce lifted rules that entail the *conclusions* and generalize w.r.t. the number of objects during testing.

2.2.4 Expressiveness and Computational Complexity

The expressive power of NLM depends on multiple factors:

1. The *depth* D of NLM (i.e., number of layers) restricts the maximum number of deduction steps.
2. The *breadth* B of NLM (i.e., the maximum number of variables in all predicates considered) limits the arity of relations among objects. Practically, most (intermediate) predicates are binary or ternary and we set B depending on the task (typically 2 or 3).
3. The number of output predicates used at each layer ($C_i^{(r)}$ in Fig. 2-3). Let $C = \max_{i,r} C_i^{(r)}$, and this number is often small in our experiments (e.g., 8 or 16).
4. In Eq. 2.2, the expressive power of MLP (number of hidden layers and number of hidden neurons) restricts the complexity of the boolean logic to be represented. In our experiments, we usually prefer shallow networks (e.g., 0 or 1 hidden layer) with a small number of neurons (e.g., 8 or 16). This can be viewed as a low-dimension regularization on the logic complexity and encourages the learned rule to be simple.

The computational complexity of NLM's forward or backward propagation is $O(m^B DC^2)$ where m is the number of objects. The network has $O(DC^2)$ parameters, which is independent of the number of objects. Assuming B is a small constant, the computational complexity of NLM is quadratic in the number of allowed predicates.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Learning to Describe Events with Temporal and Object Quantification Networks

This chapter presents *Temporal and Object Quantification Networks* (TOQ-Nets), a new class of neuro-symbolic networks with a structural bias that enables them to learn to recognize complex relational-temporal events. This is done by including reasoning layers that implement finite-domain quantification over objects and time. The structure allows them to generalize directly to input instances with varying numbers of objects in temporal sequences of varying lengths. We evaluate TOQ-Nets on input domains that require recognizing event-types in terms of complex temporal relational patterns. We demonstrate that TOQ-Nets can generalize from small amounts of data to scenarios containing more objects than were present during training and to temporal warpings of input sequences.

The material of this chapter has been previously published in Mao *et al.* [2021]. Zhezheng Luo, in particular, contributed significantly to the materials presented in this chapter.

3.1 Introduction

Every day, people interpret events and actions in terms of concepts, defined over temporally evolving relations among agents and objects [Zacks *et al.*, 2007; Stränger and Hommel, 1996]. For example, in a soccer game, people can easily recognize when one player has control of the ball, when a player *passes* the ball to another player, or when a player is *offsides*. Although it requires reasoning about intricate relationships

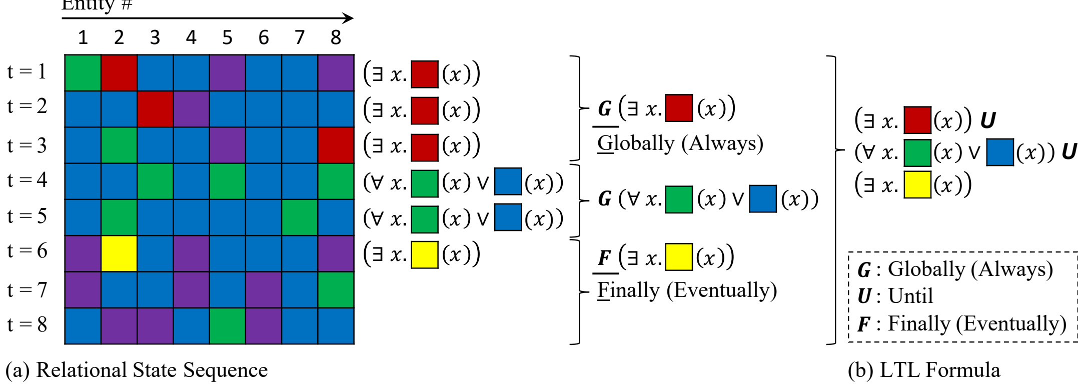


Figure 3-1: (a) An input sequence composed of relational states: each column represents the state of an entity that changes over time. A logic formula describes a complex concept or feature that is true of this temporal sequence using object and temporal quantification. The sequence is segmented into three stages: throughout the first stage, holds for at least one entity, until the second stage, in which each entity is always either or , until the third stage, in which eventually becomes true for at least one of the entities. (b) Such events can be described using first-order linear temporal logic expressions.

among sets of objects over time, this cognitive act is effortless, intuitive, and fast. It also generalizes directly over different numbers and arrangements of players, and detailed timings and trajectories. In contrast, most computational representations of sequential concepts are based on fixed windows of space and time, and lack the ability to perform relational generalization.

In this paper, we develop generalizable representations for learning complex activities in time sequences from realistic data. As illustrated in Fig. 3-1, we can describe complex events with a first-order linear temporal logic [FO-LTL; Pnueli, 1977] formula, which allows us to flexibly decompose an input sequence into stages that satisfy different criteria over time. Object quantifiers ($\forall$ and $\exists$) are used to specify conditions on sets of objects that define each stage. Such representations immediately support generalization to situations with a varying number of objects, and sequences with different time warpings.

More concretely, the variety of complicated spatio-temporal trajectories that *high pass* can refer to in a soccer game can be described in these terms: in a high pass from player A to teammate B, A is *close* to the ball ($distance(A, ball) < \theta_1$) and moving ($velocity(A) > \theta_2$) *until* the ball moves over the ground ($z_{position}(ball) > \theta_3$), which is in turn *until* teammate B gets control of the ball ($teammate(A, B) \wedge distance(B, ball) < \theta_1$). Beyond modeling human actions in physical environments, these structures can be applied to events in any time sequence of relational states, e.g., characterizing

particular offensive or defensive maneuvers in board games such as chess or in actual conflicts, or detecting a process of money-laundering amidst financial transaction records.

In this paper, we propose a neuro-symbolic approach to learning to recognize temporal relational patterns, called *Temporal and Object Quantification Networks* (TOQ-Nets), in which we design structured neural networks with an explicit bias that represents finite-domain quantification over both entities and time. A TOQ-Net is a multi-layer neural network whose inputs are the properties of agents and objects and their relationships, which may change over time. Each layer in the TOQ-Net performs either *object* or *temporal* quantification.

The key idea of TOQ-Nets is to use *tensors* to represent relations between objects, and to use tensor pooling operations over different dimensions to realize temporal and object quantifiers ($\boldsymbol{G}$ and $\forall$). For example, the colorful matrix in Fig. 3-1(a) can be understood as representing a unary color property of a set of 8 entities over a sequence of 8 time steps. Representing a sequence of relations among objects over time would require a 3-dimensional tensor. Crucially, the design of TOQ-Nets allows *the same network weights* to be applied to domains with different numbers of objects and time sequences of different lengths. By stacking object and temporal quantification operations, TOQ-Nets can easily learn to represent higher-level sequential concepts based on the relations between entities over time, starting from low-level sensory input and supervised with only high-level class labels.

There are traditional symbolic learning or logic synthesis methods that construct first-order or linear temporal logic expressions from accurate symbolic data [Neider and Gavran, 2018; Camacho *et al.*, 2018; Chou *et al.*, 2020]. TOQ-Nets take a different approach and can learn from noisy data by backpropagating gradients, which allows them to start with a general perceptual processing layer that is directly fed into logical layers for further processing.

We evaluate TOQ-Nets on two perceptually and conceptually different benchmarks: trajectory-based sport event detection and human activity recognition, demonstrating several important contributions. First, TOQ-Nets outperform both convolutional and recurrent baselines for modeling temporal-relational concepts across benchmarks. Second, by exploiting temporal-relational features learned through supervised learning, TOQ-Nets achieve strong few-shot generalization to novel actions. Finally, TOQ-Nets exhibit strong generalization to scenarios with more entities than were present during training and are robust w.r.t. time warped input trajectories. These results illustrate the power of combining symbolic representational structures with learned continuous-parameter representations to achieve robust, generalizable interpretation of complex

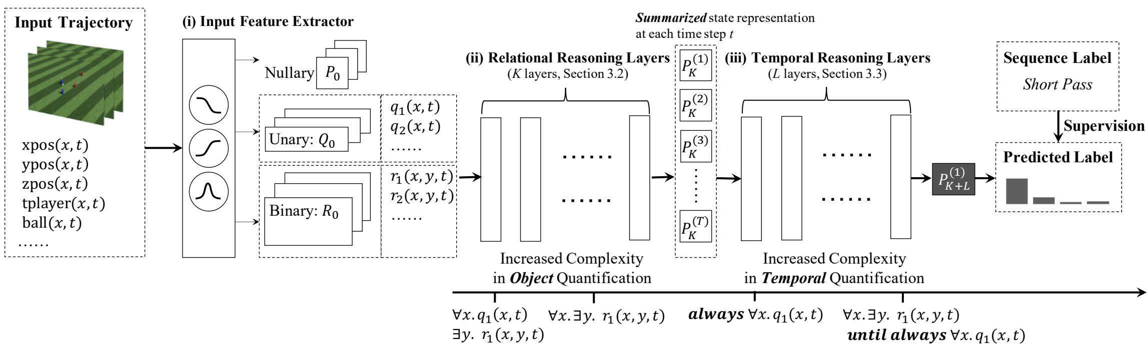


Figure 3-2: A TOQ-Net contains three modules: (i) an input feature extractor, (ii) relational reasoning layers, and (iii) temporal reasoning layers. To illustrate the model's representational power, we show that logical forms of increasing complexity can be realized by stacking multiple layers.

relational-temporal events.

3.2 TOQ-Nets

The input to a TOQ-Net is a tensor representation of the properties of all entities at each moment in time. For example, in a soccer game, the input encodes the position of each player and the ball, as well as their team membership at each step of an extended time interval. The output is a label of the category of the sequence, such as the type of soccer play it contains.

The first layer of a TOQ-Net (Fig. 3-2 (i)) extracts temporal features for each entity with an *input feature extractor* that focuses on entity features within a fixed and local time window. These features may be computed, e.g., by a convolutional neural network or a bank of parametric feature templates. The output of this step is a collection of nullary, unary, and binary relational features over time for all entities. Throughout the paper we will assume that all output tensors of this layer are binary-valued, but it can be extended directly to real-valued functions. This input feature extractor is task-specific and is not the focus of this paper.

Second, these temporal-relational features go through several *relational reasoning* layers (RRLs), detailed in Section 3.2.2, each of which performs linear transformations, sigmoid activation, and *object quantification* operations. The linear and sigmoid functions allow the network to realize learned Boolean logical functions, and the object quantification operators can realize quantifiers. Additional RRLs enable deeper nesting of quantified expressions, as illustrated in Fig. 3-2. All operations in these layers are performed for all time steps in parallel.

Next, the RRLs perform a final quantification, computing for each time step a set of a nullary features that are passed to the *temporal reasoning layers* (TRLs), as detailed in Section 3.2.3. Each TRL performs linear transformations, sigmoid activation, and *temporal quantification*, allowing the model to realize a subset of linear temporal logic [Pnueli, 1977]. As with RRLs, adding more TRLs enables the network to realize logical forms with more deeply nested temporal quantifiers.

In the last layer, all object and time information is projected into a set of features of the initial time step, which summarize the temporal-relational properties of the entire trajectory (e.g., "the kicker eventually scores"), and fed into a final softmax unit to obtain classification probabilities for the sequence.

It is important to understand the representational power of this model. The *input transformation layer* learns basic predicates and relations that will be useful for defining more complex concepts, but no specific predicates or relations are built into the network in advance. The relational reasoning layers build quantified expressions over these basic properties and relations, and might construct expressions that could be interpretable as "the player is close to the ball." Finally, the temporal reasoning layer applies temporal operations to these complex expressions, such as "the player is close to the ball until the ball moves with high speed." Critically, *none* of the symbolic properties or predicates are hand defined—they are all constructed by the initial layer in order to enable the network to express the concept it is being trained on.

TOQ-Nets are not fully first-order: all quantifiers operate only over the finite domain of the input instance, and can be seen as "short-hand" for finite disjunctions or conjunctions over objects or time points. In addition, the depth of the logical forms it can learn is determined by the fixed depth of the network. However, our goal is not to fully replicate temporal logic, but to bring ideas of object and temporal quantification into neural networks, and to use them as structural inductive biases to build models that generalize better from small amounts of data to situations with varying numbers of objects and time courses.

3.2.1 Temporal-Relational Feature Representation

TOQ-Nets use tensors as internal representations between layers; they represent, all at once, the values of all predicates and relations grounded on all objects at all time points. The operations in a TOQ-Net are vectorized, operating in parallel on all objects and times, sometimes expanding the dimensionality via outer products, and then re-projecting into smaller dimensions via max-pooling. This processing style is analogous to representing an entire graph using an adjacency matrix and using matrix operations to compute properties of the nodes or of the entire graph. In TOQ-Nets,

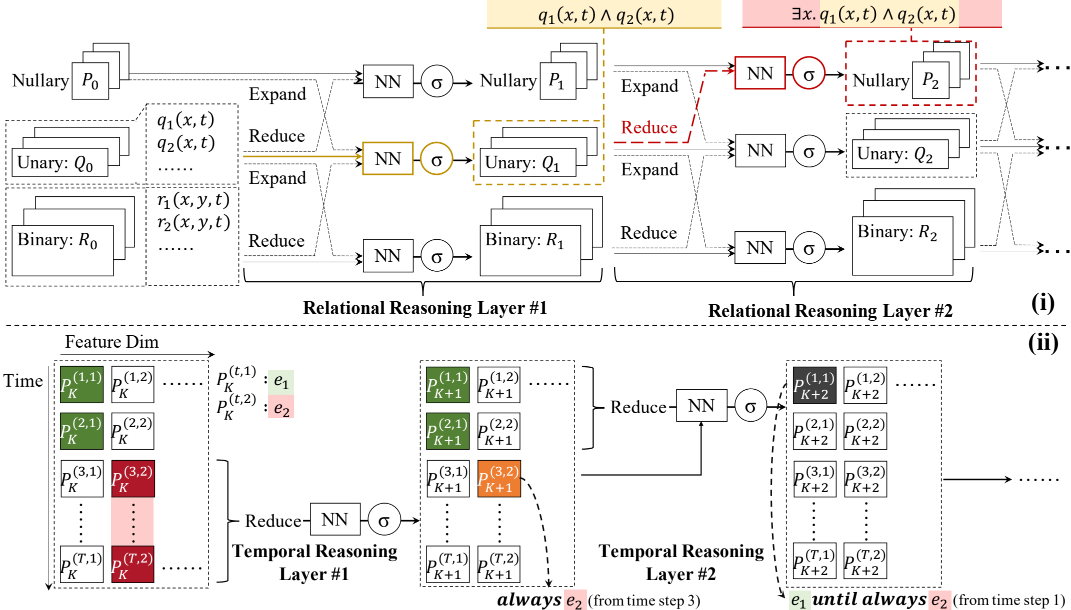


Figure 3-3: Illustration of (i) relational reasoning layers and (ii) temporal reasoning layers. We provide two illustrative running traces. (i) The first relational reasoning layer takes unary predicates q_1 and q_2 as input and its output Q_1 is able to represent $q_1 \wedge q_2$. The $\max(Q_1, dim = 0)$ in layer 2 can represent $\exists x.\ q_1(x,t) \wedge q_2(x,t)$. (ii) Assume P_K encodes the occurance of events e_1 and e_2 at each time step. The first temporal reasoning layer can realize *always* e_2 with a temporal pooling from time step 3 to time step T. In the second temporal reasoning layer, the temporal pooling summarizes that e_1 holds true from time step 1 to 2. Thus, the NN should be able to realize e_1 *until* (*always* e_2).

the input to the network, as well as the feature output of intermediate layers, is represented as a tuple of three tensors.

Specifically, we use a vector of dimension D_0 to represent aspects of the state that are global and do not depend on any specific object at each time t. We use a matrix of shape $N \times D_1$ to represent the unary properties of each entity at time t, where N is the number of entities and D_1 is the hidden dimension size. Similarly, we use a tensor of shape $N \times N \times D_2$ to represent the relations between each pair of entities at time step t. As a concrete example, illustrated in Fig. 3-2, the number of entities N is the total number of players plus one (the ball). For each entity x and each time step, the inputs are their 3D position, type (ball or player) and team membership. The TOQ-Net outputs the action performed by the target player. Since there are only entity features, the input trajectory is encoded with a "unary" tensor of shape $T \times N \times D_1$, where T is the length of the trajectory. That is, there are no nullary or binary inputs in this case.

3.2.2 Relational Reasoning Layers

Our Relational reasoning layers (RRLs) follow prior work on Neural Logic Machines [Dong *et al.*, 2019], illustrated in Fig. 3-3 (i). Consider a specific time step t. At each layer l, the input to a neural logic layer is a 3-tuple $(P_{l-1}, Q_{l-1}, R_{l-1})$, which corresponds to nullary, unary, and binary features respectively. Their shapes are D_0, $N \times D_1$, and $N \times N \times D_2$. The output is another 3-tuple (P_l, Q_l, R_l), given by

$$\begin{aligned}
P_l &= \mathrm{NN}_P \left(\mathrm{Concat}\left[P_{l-1}; \max(Q_{l-1}, \dim = 0) \right] \right), \\
Q_l &= \mathrm{NN}_Q \left(\mathrm{Concat}\left[Q_{l-1}; \max(R_{l-1}, \dim = 0); \right. \right. \\
&\qquad \left. \left. \max(R_{l-1}, \dim = 1); \mathrm{expand}(P_{l-1}, \dim = 1) \right] \right), \\
R_l &= \mathrm{NN}_R \left(\mathrm{Concat}\left[R_{l-1}; \mathrm{expand}(Q_{l-1}, \dim = 0); \right. \right. \\
&\qquad \left. \left. \mathrm{expand}(Q_{l-1}, \dim = 1) \right] \right).
\end{aligned}$$

where NN_* are single fully-connected layers with sigmoid activations. For unary and binary features, NN_Q and NN_R are applied along the feature dimension. That is, we apply the same linear transformation to the unary features of all entities. A different linear transformation is applied to the binary features of all entity pairs. $\mathrm{Concat}[\cdot\ ;\ \cdot]$ is the concatenation operation, applied to the last dimension of the tensors (the feature dimension). max, also called the "reduced" max operation, takes the maximum value along the given axis of a tensor. The expand operation, also called "broadcast," will duplicate the input tensor N times and stack them together along the given axis. RRLs are applied identically to the input features at every time step t. That is, we use the same neural network weights in a RRL for all time steps in parallel.

RRLs are motivated by relational logic rules in a finite and fully grounded universe. The max reduction operations implement a differentiable version of an existential quantifier over the finite universe of individuals, given that the truth values of the propositions are represented as values in $(0.0, 1.0)$. Because preceding and subsequent RRLs can negate propositions as needed, we omit explicit implementation of finite-domain universal quantification, although it could be added by including analogous min reductions. Thus, as illustrated in Fig. 3-3 (i), given input features $q_1(x, t)$ and $q_2(x, t)$, we can realize the formula $\exists x.\ q_1(x, t) \wedge q_2(x, t)$ by stacking two such layers.

Throughout the paper we have been using only nullary, unary, and binary features, but the proposed framework itself can be easily extended to higher-order relational features. From a graph network [Bruna *et al.*, 2014; Kipf and Welling, 2017; Battaglia *et al.*, 2018] point of view, one can treat these features as the node and edge embeddings of a fully-connected graph and the relational reasoning layers as specialized graph

neural network layers for realizing object quantifiers.

3.2.3 Temporal Reasoning Layers

Temporal reasoning layers (TRLs) perform *quantification* operations similar to relational reasoning layers, but along the *temporal* rather than the object dimension. The first TRL takes as input the summarized event representation produced by the K-th relational reasoning layer, $P_K^{(t)}$ for at all time steps t, as a matrix of shape $T \times D$. Each TRL is computed as

$$P_{K+l}^{(t)} = \max_{t'>t} \mathrm{NN}_l \left(\mathrm{Concat} \left[P_{K+l-1}^{(t')}; \max_{t \le t'' < t'} P_{K+l-1}^{(t'')} \right] \right). \tag{3.1}$$

We will walk through the subexpressions of this formula.

1. P_{K+l-1} is the output tensor of the previous temporal reasoning layer, of shape $T \times C$, where T is the number of time steps, and C is the number of feature channels. Each entry in this tensor $P_{K+l-1}^{(t)}[i]$ can be interpreted as: event i happens at time t.

2. $\left(\max_{t \le t'' < t'} P_{K+l-1}^{(t'')}\right)$, abbreviated as $Q_{k+l-1}^{(t,t')}$ in the following text, is a vector of shape C. Its entry $Q_{k+l-1}^{(t,t')}[i]$, where $t \le t'$, represents the concept that event i happens some time between t and t'. **Importantly,** together with the preceding and subsequent neural network operations, which can realize negation operations, it also allows us to describe the event that i holds true for all time steps between t and t'.

3. NN_l is a fully connected neural network with sigmoidal activation, which gets uniformly applied to all time steps t and a future time step $t' > t$. Its input is composed of two parts: the events that happen at t', i.e., $P_{K+l-1}^{(t)}$, and the events that happen between t and t', summarized with temporal quantification operations, i.e., $\left(\max_{t \le t'' < t'} P_{K+l-1}^{(t'')}\right)$.

4. The outer-most max pooling operation $\max_{t'>t}$ enumerates over all $t' > t$, and test whether the condition specified by NN_l holds for at least one such t'.

A special case is the first temporal reasoning layer. It takes P_K as its input, which is the output of last relational reasoning layer. Thus, the first temporal reasoning layer implements:

$$P_{K+1}^{(t)} = \mathrm{NN}_1 \left(\max_{t \le t'' < T} P_K^{(t'')} \right),$$

t	Input		1-st Layer				2-nd Layer
	$p(t)$	$q(t)$	$\boldsymbol{G}p$	$\boldsymbol{F}p$	$\boldsymbol{G}q$	$\boldsymbol{F}q$	$p\ \boldsymbol{U}\ (\boldsymbol{G}q)$
1	T	T	F	T	F	T	T
2	T	F	F	T	F	T	T
3	F	T	F	F	T	T	F
4	F	T	F	F	T	T	F

Table 3.1: A running example of different temporal quantification formulas that TOQ-Nets can realize. For clarity, we use T and F for True/False. In the actual computation, they are "soft" Boolean values ranges in $[0, 1]$. **G** means ***always***; **F** means ***Eventually***; **U** means ***until***.

Algorithm 1 An example temporal structure that the second temporal reasoning layer can recognize.

Input: $p(t)$, $q(t)$, $(\boldsymbol{G}p)(t)$, $(\boldsymbol{F}p)(t)$, $(\boldsymbol{G}q)(t)$, and $(\boldsymbol{F}q)(t)$
Output: $(p\,\boldsymbol{U}(\boldsymbol{G}q)))\,(t)$, which is *true* if p holds *true* from time step t until q becomes always *true*.
1: **for** $t \leftarrow 1$ to T **do**
2:     **for** $t' \leftarrow t+1$ to T **do**
3:         **if** $\forall t'' \in [t, t').\ p(t'')$ and $(\boldsymbol{G}q)(t')$ **then**
4:             $(p\,\boldsymbol{U}(\boldsymbol{G}q)))\,(t) \leftarrow$ *true*
5:         **end if**
6:     **end for**
7: **end for**

---

where T is the sequence length. Note that there is no enumeration for a future time step $t' > t$ involved. In addition, for all temporal layers, we add residual connections by concatenating their inputs with the outputs.

Next, let's consider a running example illustrating how a TOQ-Net can recognize the event that: event p holds true until event q becomes always true. In LTL, this can be written as $p\,\boldsymbol{U}(\boldsymbol{G}q)$. Using the plain first-order logic (FOL) language, we can describe it as: $\exists\, t. [\forall t'. \ (0 \le t' < t) \implies p(t')) \wedge (\forall t'. \ (t' \ge t) \implies q(t')]$.

For simplicity, we consider a tensor representation for two events $p(t)$ and $q(t)$, where $p(t) = 1$ if it happens at time step t and $p(t) = 0$ otherwise. Given the input sequence of length 4 in Table 3.1 ($p(t)$ and $q(t)$), the first layer is capable of computing the following four properties for each time step t: $\boldsymbol{G}p$ (***always*** p), which is true if p holds true for all future time steps starting from t, $\boldsymbol{F}p$ (***eventually*** p), which is true if p is true for at least one time step starting from t, and similarly, $\boldsymbol{G}q$ (***always*** q) and $\boldsymbol{F}q$ (***eventually*** q). Overall, together with residual connections, the first layer can recognize six useful events: $p(t)$, $q(t)$ (from residual connection), $\boldsymbol{G}p$, $\boldsymbol{F}p$, $\boldsymbol{G}q$, and $\boldsymbol{F}q$ (by temporal quantification, i.e. pooling operations along the temporal dimension).

The second layer can realize the computation depicted in Algorithm 1. For every time step t, it enumerates all $t' > t$ and computes the output based on

1. the events at t' (represented as $P_{K+l-1}^{(t')}$ in Equation 3.1, concretely the $(\boldsymbol{G}q)(t')$ in the Algorithm 1 example), and
2. the state of another event between t and t' (represented as $\left(\max_{t \le t'' < t'} P_{K+l-1}^{(t'')}\right)$ in Equation 3.1, concretely the $\forall t'' \in [t, t').\ p(t'')$ in the Algorithm 1 example).

From the perspective of First-Order Linear Temporal Logic (FO-LTL), stacking multiple temporal reasoning layers enables us to realize FO-LTL formulas such as:

$$p_1 \ \boldsymbol{U}\, p_2 \ \boldsymbol{U}\, p_3 \ \boldsymbol{U} \ \cdots \ \boldsymbol{U}\, p_k,$$

which is interpreted as p_1 holds true until p_2 becomes true and p_2 holds true from that until p_3 becomes true $\cdots$, and

$$\boldsymbol{F} p_1 \ \boldsymbol{XF}\, p_2 \ \boldsymbol{XF}\, p_3 \ \boldsymbol{XF} \ \cdots \ \boldsymbol{XF}\, p_k,$$

which is interpreted as p_1 eventually becomes true and after that p_2 eventually becomes true and after that $\cdots$, and in addition, formulas with interleaved until and eventually quantifiers. Here, $\boldsymbol{XF}$ is a composition of the ne$\boldsymbol{X}$t operator and the $\boldsymbol{F}$inally operator in LTL. Meanwhile, as described so far, TOQ-Nets can only nest object quantification inside temporal quantification, so it can represent ***always*** $\exists x.\ q_1(x) \wedge q_2(x)$, but not $\exists x.$ ***always*** $q_1(x) \wedge q_2(x)$. This can be solved by interleaving relational and temporal reasoning layers.

It is important to notice that, by using object and temporal pooling operations together with trainable neural networks to realize logic formulas with object and temporal quantifiers, the idea itself generalizes to a broader set of FO-LTL formulas. We design TOQ-Nets to model only a subset of FO-LTL formulas, because they can be computed efficiently (with only $O(T^2)$ space) and they are expressive enough for the type of data we are trying to model.

3.3 Experiments

We compare our model with other approaches to object-centric temporal event detection and to concept learning over robot object-manipulation trajectories in this section. The setups and metrics focus on data efficiency and generalization.

Model	Reg.	Few-Shot	Full
STGCN	$73.2_{\pm 1.6}$	$26.0_{\pm 5.7}$	$62.8_{\pm 0.6}$
STGCN-MAX	$73.6_{\pm 1.5}$	$28.6_{\pm 5.0}$	$63.6_{\pm 0.7}$
STGCN-LSTM	$72.7_{\pm 1.4}$	$23.8_{\pm 5.9}$	$61.9_{\pm 0.6}$
Space-Time	$74.8_{\pm 1.5}$	$31.7_{\pm 6.1}$	$65.2_{\pm 0.6}$
Non-Local	$76.5_{\pm 2.4}$	$45.0_{\pm 6.3}$	$69.5_{\pm 2.4}$
TOQ-Net (ours)	$\mathbf{87.7}_{\pm 1.3}$	$\mathbf{52.2}_{\pm 6.3}$	$\mathbf{79.8}_{\pm 0.8}$

Table 3.2: Results on the soccer event dataset. Different columns correspond to different action sets (the regular, few-shot, and full action sets). The performance is measured by per-action (macro) accuracy, averaged over nine few-shot splits. The $\pm$ values indicate standard errors. TOQ-Net significantly outperforms all baseline methods on the few-shot action set.

3.3.1 Baseline Approaches

We compare TOQ-Nets against five baselines. The first two are spatial-temporal graph convolutional neural networks [Yan *et al.*, 2018, STGCN;] and its variant STGCN-MAX, which models entity relationships with graph neural networks and models temporal structure with temporal-domain convolutions. The third is STGCN-LSTM, which uses STGCN layers for entity relations but LSTM [Hochreiter and Schmidhuber, 1997] for temporal structures. The last two baselines are based on space-time graphs: Space-Time Graph [Wang and Gupta, 2018] and Non-Local networks [Wang *et al.*, 2018]. We provide details about our implementation and how we choose the model configurations in Appendix A.1.

3.3.2 Trajectory-Based Soccer Event Detection

We start our evaluation with an event-detection task in soccer games. The task is to recognize the action performed by a specific player at specific time step in a soccer game trajectory.

Dataset and setup. We collect training and evaluation datasets based on the gfootball simulator*, which provides a physics-based 3D football simulation. It also provides AI agents that can be used to generate random plays. The simulator provides the 3D coordinates of the ball and the players as well as the action each player is performing at each time step. There are in total 13 actions defined in the simulator, including *movement, ball_control, trap, short_pass, long_pass, high_pass, header, shot, deflect, catch, interfere, trip* and *sliding*. We exclude *header* and *catch* actions, as they

*https://research-football.dev/

never appear in AI games. We also exclude *ball_control* and *movement*, since they just mean the agent is moving (with or without the ball). Thus, in total, we have nine action categories. We run the simulator with AI-controlled players to generate plays, and formulate the task as classifying the action (9-way classification) of a specific player at a specific time step given a temporal context (25 frames). For each action, we have generated 5,000 videos, expect for *sliding*, for which we generated 4,000 videos because it is rare in the AI games. Among the generated examples, 62% (2,462 or 3,077) are used for training, 15% are used for validation, and 23% are used for testing. Each trajectory is an 8-fps replay clip that contains 17 frames (about two seconds). There is a single "target" player in each trajectory. The action label of the trajectory is the action performed by this target player at frame #9. We randomly split all actions into two categories: seven "regular" actions, for which all game plays are available, and two "few-shot" actions, for which only 50 clips are available during training.

Input features. Each trajectory is represented with 7 time-varying unary predicates, including the 3D coordinate of each player and the ball and four extra predicates defining the type of each entity x: IsBall(x), IsTargetPlayer(x), SameTeam(x), OpponentTeam(x), where SameTeam(x) and OpponentTeam(x) indicates whether x is of the same team as the target player. We also add a temporal indicator function which is a Gaussian function centered at frame of interest with variance $\sigma^2 = 25$.

Results. Table 3.2 shows the result. Our model significantly outperforms all the baselines in all three action settings, suggesting that our model is able to discover a set of useful features at both input and intermediate levels and use them to compose new action classifiers from only a few examples.

Generalization to more players. Due to their object-centric design, TOQ-Nets can generalize to soccer games with a varying number of agents. After training on 6v6 soccer games (i.e., 6 players on each team), we evaluate the performance of different models on games with different numbers of players: 3v3, 4v4, 8v8, and 11v11. For each action we have generated, on average, 1,500 examples for testing. Table 3.3 summarizes the result. Comparing the columns highlighted in yellow, we notice a significant performance drop for all baselines while TOQ-Net performs the best. By visualizing data and predictions, we found that misclassifications of instances of *shot* as *short pass* contribute most to the performance degradation of our model when we have more players. Specifically, the recall of *shot* drops from 97% to 60%. In soccer

Model	Time Warp	3v3	4v4	6v6	8v8	11v11
STGCN	N	$40.7_{\pm 1.0}$ (-40.4%)	$63.2_{\pm 4.9}$ (-7.4%)	$68.2_{\pm 2.8}$ (0.0%)	$55.4_{\pm 3.3}$ (-18.8%)	$44.4_{\pm 2.1}$ (-34.9%)
STGCN	Y	$32.6_{\pm 2.8}$ (-52.3%)	$50.2_{\pm 4.4}$ (-26.5%)	$52.8_{\pm 7.0}$ (-22.6%)	$43.2_{\pm 4.9}$ (-36.7%)	$34.0_{\pm 3.7}$ (-50.2%)
STGCN-MAX	N	$47.4_{\pm 3.2}$ (-33.7%)	$68.8_{\pm 2.0}$ (-3.8%)	$71.5_{\pm 1.9}$ (0.0%)	$59.1_{\pm 0.7}$ (-17.3%)	$45.6_{\pm 2.5}$ (-36.2%)
STGCN-MAX	Y	$37.5_{\pm 7.2}$ (-47.5%)	$52.3_{\pm 4.2}$ (-26.9%)	$56.5_{\pm 4.5}$ (-21.0%)	$46.6_{\pm 3.7}$ (-34.8%)	$36.9_{\pm 1.9}$ (-48.4%)
STGCN-LSTM	N	$39.7_{\pm 1.1}$ (-43.1%)	$60.4_{\pm 0.2}$ (-13.5%)	$69.8_{\pm 0.1}$ (0.0%)	$55.8_{\pm 2.0}$ (-20.0%)	$44.1_{\pm 0.7}$ (-36.8%)
STGCN-LSTM	Y	$21.8_{\pm 0.8}$ (-68.8%)	$27.8_{\pm 1.3}$ (-60.2%)	$30.6_{\pm 0.6}$ (-56.1%)	$25.8_{\pm 1.0}$ (-63.1%)	$22.6_{\pm 0.8}$ (-67.6%)
Space-Time	N	$29.0_{\pm 1.6}$ (-60.4%)	$53.5_{\pm 3.2}$ (-27.0%)	$73.3_{\pm 0.3}$ (0.0%)	$33.9_{\pm 2.8}$ (-53.7%)	$15.2_{\pm 1.8}$ (-79.3%)
Space-Time	Y	$29.7_{\pm 3.1}$ (-59.5%)	$51.6_{\pm 2.5}$ (-29.6%)	$70.7_{\pm 0.3}$ (-3.5%)	$33.8_{\pm 2.2}$ (-53.9%)	$14.9_{\pm 1.6}$ (-79.7%)
Non-Local	N	$45.9_{\pm 5.1}$ (-41.2%)	$70.7_{\pm 5.3}$ (-9.5%)	$78.1_{\pm 5.8}$ (0.0%)	$58.5_{\pm 10.8}$ (-25.1%)	$41.8_{\pm 13.6}$ (-46.5%)
Non-Local	Y	$46.7_{\pm 4.1}$ (-40.2%)	$69.9_{\pm 4.7}$ (-10.5%)	$77.7_{\pm 5.0}$ (-0.5%)	$58.7_{\pm 12.6}$ (-24.9%)	$41.3_{\pm 13.6}$ (-47.1%)
TOQ-Net	N	$\mathbf{77.4}_{\pm 3.5}$ (-12.4%)	$\mathbf{88.3}_{\pm 0.7}$ (-0.0%)	$\mathbf{88.4}_{\pm 0.6}$ (0.0%)	$81.3_{\pm 1.7}$ (-8.0%)	$\mathbf{77.1}_{\pm 1.7}$ (-12.8%)
TOQ-Net	Y	$76.0_{\pm 2.6}$ (-14.0%)	$87.7_{\pm 1.4}$ (-0.8%)	$\mathbf{86.9}_{\pm 0.4}$ (-1.7%)	$80.3_{\pm 1.1}$ (-9.1%)	$74.9_{\pm 2.3}$ (-15.2%)

Table 3.3: Full results on generalization to scenarios with all combinations of #agents and temporally warping on the soccer event dataset. The standard error of all values are smaller than 2.5%, computed based on three random seeds.

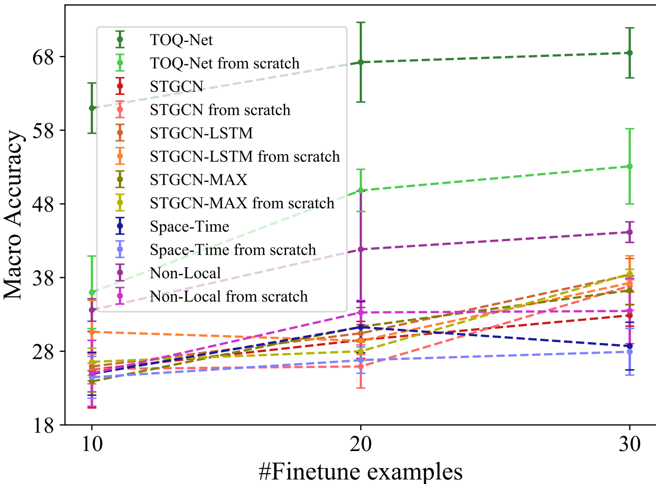


Figure 3-4: Generalization to soccer environments with a different court size and agent speeds. The standard errors are computed based on three random seeds.

plays with many agents, a shot is usually unsuccessful and a player from another team steals the ball in the end. In such scenarios, TOQ-Net tends to misclassify such trajectories as a *short pass*. Ideally, this issue should be addressed by understanding actions based on agents' goals instead of the actual outcome [Intille and Bobick, 2001]. We leave this extension as a future direction.

Generalization to temporally warped trajectories. Another crucial property of TOQ-Nets is to recognize actions based on their sequential order in the input trajectory, instead of binding features to specific time steps in the trajectory. To show this, we test the performance of different models on time warped trajectories. Each test trajectory has a length of 25, and each trajectory is labeled by the action performed by the target player at any time step between the 6-th and 19-th frame. We ensure that the target player performs only one action during the entire input trajectory. Thus, the label is unambiguous. The results are shown in Table 3.3. Specifically, our test set consists of 25-frame trajectories, and the action may occur at anytime between the 6th and the 19th frame. By comparing rows with and without time warping, we notice a 60% performance drop for STGCN, STGCN-MAX, and STGCN-LSTM. In contrast, TOQ-Nets still have reasonable performance. Note that Space-Time and Non-Local model have almost no performance drop against time warping because they are completely agnostic to temporal ordering.

Generalization to different input scales Intuitively, after learning how to detect events in a soccer game, the learner should also generalize concepts to analogs with

Model	#params	Reg.	Few-Shot	Full
STGCN	3.42M	$72.0_{\pm 1.5}$	$22.6_{\pm 5.2}$	$61.0_{\pm 0.8}$
STGCN_S	263K	$73.0_{\pm 1.5}$	$22.5_{\pm 4.9}$	$61.7_{\pm 0.5}$
$\textbf{STGCN}_T$	34K	$73.2_{\pm 1.6}$	$26.0_{\pm 5.7}$	$\mathbf{62.8}_{\pm 0.6}$
STGCN-LSTM	2.08M	$72.5_{\pm 1.6}$	$20.2_{\pm 5.0}$	$60.9_{\pm 0.7}$
STGCN-LSTM_S	233K	$72.1_{\pm 1.7}$	$20.5_{\pm 5.9}$	$60.6_{\pm 0.7}$
$\textbf{STGCN-LSTM}_T$	33K	$72.7_{\pm 1.4}$	$23.8_{\pm 5.9}$	$\mathbf{61.9}_{\pm 0.6}$
STGCN-MAX	3.42M	$73.5_{\pm 4.8}$	$20.2_{\pm 6.0}$	$61.7_{\pm 4.1}$
$\textbf{STGCN-MAX}_S$	263K	$73.6_{\pm 1.5}$	$28.6_{\pm 5.0}$	$\mathbf{63.6}_{\pm 0.7}$
STGCN-MAX_T	34K	$74.5_{\pm 1.2}$	$25.4_{\pm 6.1}$	$63.6_{\pm 0.7}$
Space-Time	263K	$71.3_{\pm 1.4}$	$26.9_{\pm 7.1}$	$61.4_{\pm 1.2}$
Space-Time_S	24K	$74.7_{\pm 1.4}$	$30.7_{\pm 6.8}$	$64.9_{\pm 0.6}$
$\textbf{Space-Time}_T$	14K	$74.8_{\pm 1.5}$	$31.7_{\pm 6.1}$	$\mathbf{65.2}_{\pm 0.6}$
Non-Local	1.23M	$74.5_{\pm 4.2}$	$44.3_{\pm 7.0}$	$67.8_{\pm 3.7}$
$\textbf{Non-Local}_S$	108K	$76.5_{\pm 2.4}$	$45.0_{\pm 6.3}$	$\mathbf{69.5}_{\pm 2.4}$
Non-Local_T	32K	$75.6_{\pm 1.2}$	$44.8_{\pm 6.4}$	$68.8_{\pm 1.0}$
TOQ-Net	35K	$87.7_{\pm 1.3}$	$52.2_{\pm 6.3}$	$79.8_{\pm 0.8}$

Table 3.4: Results on the soccer event dataset for baselines with different capacities, where model_S and model_T denote the small and the tiny variant for each model. The performance is measured by per-action (macro) accuracy, averaged over nine few-shot splits. The $\pm$ values indicate standard errors. For each baseline we showed the best performance over three levels of capacities. In fact, performances of most of the models are not affected much by the capacity. We highlight the best-performing variants of all baselines, and use them in all comparisons in the main text.

the same basic structure but enacted on proportionately larger or smaller spatial or temporal scales—as in the soccer-inspired game futsal. To evaluate generalization to different input scales, we first train all models on the original dataset with 9-way classification supervision. After that, we finetune each model on a new dataset where all input values are doubled (i.e., the court is now double the size, and the players now move at four times the speed), but with only a small number of examples. Thus, time flows twice fast, and the player moves four times the speed compared with the original dataset. Results are summarized in Fig. 3-4. While all baselines can benefit from the pre-training on the original dataset, our TOQ-Net outperforms all baselines by a significant margin when the finetuning dataset is small.

Ablation Study: Network Size In general, TOQ-Net has a smaller number of weights than baselines. We add additional comparisons to models of different #params. Specifically, model$_S$ is a smaller version of the model and model$_T$ is the tiny version of the model whose #params are about equal to or smaller than #params of TOQ-Net. To obtain smaller models we typically reduce the number of layers and the hidden state dimensions. On the few-shot task of the soccer event dataset, we test all baselines with all combination of features and capacities. Table 3.4 summarizes the results. In general, the "small" variations are the best for all models (except for STGCN-LSTM). Across all experiments presented in the main paper, we use the best architecture in Table 3.4.

Ablation Study: #Layers of TOQ-Net We study the TOQ-Net performance over varying number of relation reasoning layers and temporal reasoning layers on the 9-way classification task of the soccer event dataset (See figure Fig. 3-5), and decide to use the combination of 3 relational layers and 4 temporal layers. Both relational layers and temporal layers play important roles in the performance. As a generalization test, we also visualize the accuracy on temporally warped trajectories. We can see that temporal reasoning layers are important for the robustness against temporal warping.

Interpretability of TOQ-Net Though interpretability is not the main focus of our paper, in Fig. 3-6, we show how TOQ-Net composes low-level concepts into high-level concepts: from inputs, to intermediate features at different temporal layers, and the final output labels. See the figure caption for detailed analysis.

3.3.3 Manipulation Concept Learning from 6D Poses

Structural action representations can also be usefully applied to other domains. Here we show the result in a robotic environment, where the goal is to classify the action performed by a robotic arm.

Dataset and setup. We generated a dataset based on the RLBench simulator [James *et al.*, 2020], which contains a set of robotic object-manipulation actions in a tabletop environment. We use 24 actions from the dataset, including *CloseBox, CloseDrawer, CloseFridge, CloseGrill, CloseJar, CloseLaptopLid, CloseMicrowave, GetIceFromFridge, OpenBox, OpenFridge, OpenMicrowave, OpenWineBottle, PickUpCup, PressSwitch, PushButtons, PutGroceriesInCupboard, PutItemInDrawer, PutRubbishInBin, PutTrayInOven, ScoopWithSpatula, SetTheTable, SlideCabinetOpenAndPlaceCups, TakePlateOffColoredDishRack,* and *TakeToiletRollOffStand*. We randomly

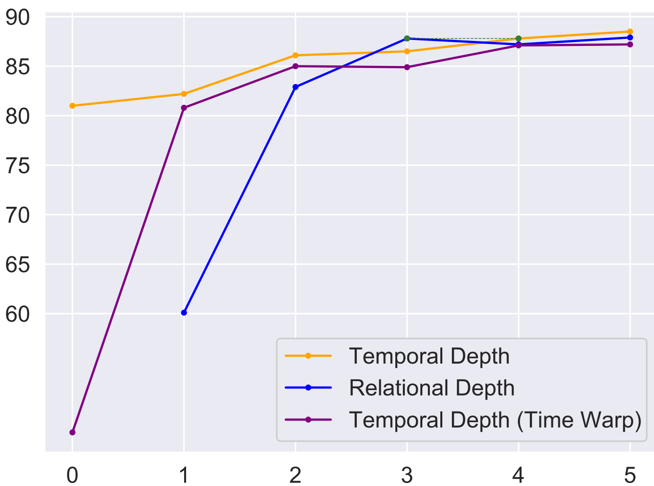


Figure 3-5: Comparing # of reasoning layers. Accuracy is tested on 6v6 9-way classification. When # of relational reasoning layers vary (blue), temporal reasoning layers are fixed at 4. When # of temporal reasoning layers vary (orange), relational reasoning layers are fixed at 3. When we have 0 temporal reasoning layers, we predict the sequence label based on the feature of the frame of interest. The purple line shows the performance on temporally warped trajectories when we have the # of relational reasoning layers fixed and vary the # of temporal reasoning layers. Relational layers are required for the prediction task (# of relational reasoning layer must be greater than 0), because there is no nullary feature input to the network: we need at least one relational reasoning layer to gather information across the entities. Overall, adding reasoning layers improves results, and 3 relational layers + 4 temporal layers is a good balance between computation and performance.

initialize the position and orientation of different objects in the scene and use the built-in motion planner to generate robot arm trajectories. Depending on the task and the initial poses of different objects, trajectories may have different lengths, varying from 30 to 1,000. Most of the trajectories have $\sim$50 frames.

For each action category, we have generated 100 trajectories. Among the generated examples, 60% of the examples are used for training, 15% are used for validation, and the other 25% are used for testing.

Input. Each trajectory contains the poses of different objects, at each time step. Each object except for the robot arm is represented as 13 unary predicates, including the 3D position, the 3D orientation represented as quaternions (4D vector), and the 3D bounding box (axis-aligned, 6D vector). The robot arm is represented as 8 unary predicates, including the 3D position, the 3D orientation, and a binary predicate that indicates whether the gripper is open or closed. Note that deformable objects are split into several pieces so that the learner can infer their state (open or closed). For

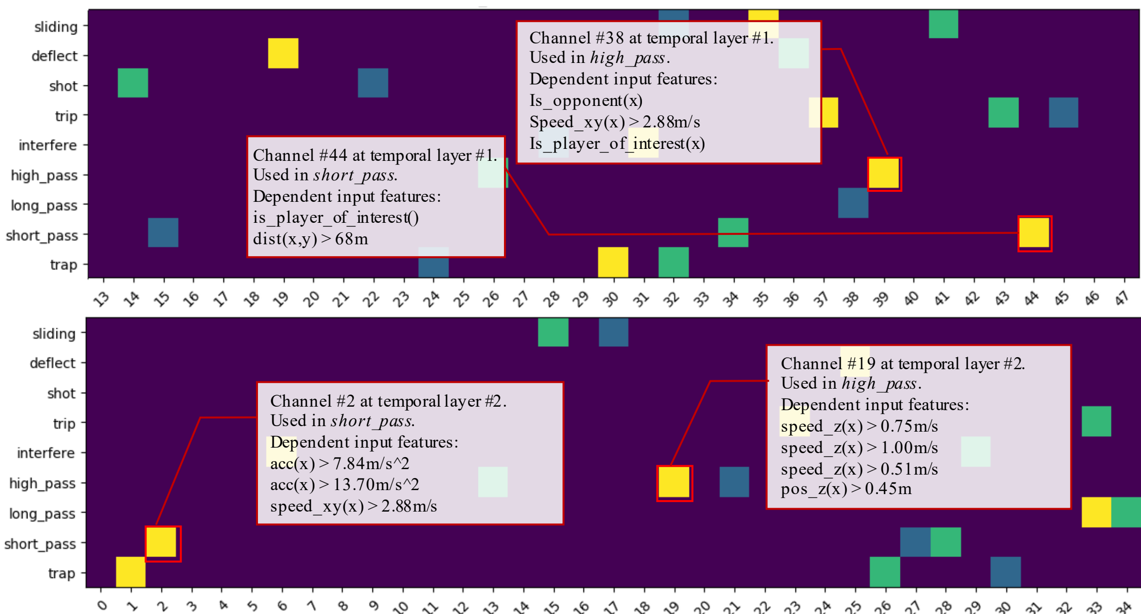


Figure 3-6: Relevant features in temporal layers. Feature dependencies are computed by gradient. These dependencies and thresholds are learned end-to-end from data. Insets detail features for events in the first and the second stage of the *high pass* and *short pass*.

example, the lid of the box is a separate object. We also input the 6D pose and the state of the robot gripper. During training, we used on average 65 robot trajectories for regular actions.

Input features. All methods take the raw trajectories as input. In TOQ-Nets, we use an extra single linear layer with sigmoid activation as the input feature extractor. To evaluate temporal modeling with recurrent neural networks, STGCN-LSTM uses kernel size 1 in its (STGCN) temporal convolution layers. More ablation studies about baselines can be found in Appendix A.1.

Results. We start with evaluating different models on the standard 24-way classification task. We summarize results in Fig. 3-7. It also illustrates each model's performance under different time stretching factors of the input trajectory, from $1/8\times$ to $8\times$. Similar to the soccer dataset, the TOQ-Net outperforms both convolutional and recurrent models across all time stretching factors.

Generalization to new action concepts. We also evaluate how well different models can generalize the learned knowledge about opening and closing other objects to novel objects. Specifically, we hold out 50% of all *Open X* actions and 50% of

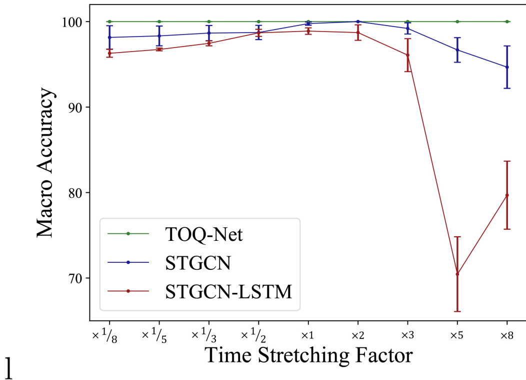
Model	Reg.	1-Shot	Full
STGCN	99.89$_{\pm 0.05}$	94.92$_{\pm 1.03}$	98.79$_{\pm 0.23}$
STGCN-LSTM	**99.92**$_{\pm 0.03}$	95.48$_{\pm 1.67}$	**98.86**$_{\pm 0.43}$
TOQ-Net	**99.96**$_{\pm 0.02}$	**98.04**$_{\pm 0.97}$	**99.48**$_{\pm 0.24}$

Table 3.5: Few-shot learning on the RLBench dataset, measured by per-action (macro) accuracy and averaged of four 1-shot splits and four random seeds per split. The $\pm$ values indicate standard errors. On the right we shows the sampled performance of different models on each individual 1-shot split.

the *Close X* actions to form the 1-shot action learning set. For example, the learner may only see a single instance of the opening of a box during training. Actions with more examples are "regular." All models are tested on a 3-way classification task: *Open-X*, *Close-X*, and *Other*. Results are summarized in Table 3.5. There are several held-out sets that are noticeably more "difficult" than others , such as the Set #1 in Table 3.5. Our visualization shows that actions *close jar* and *close drawer* are harder generalization targets, compared to other actions. This is possibly because the motion of *close jar* involves some rotation, and the motion of *close drawer* looks more like pushing than other *close-X* actions.

3.3.4 Extension to Real-World Datasets

The proposed TOQ-Net can also be extended to other real-world datasets. These examples further illustrate the robustness of TOQ-Net to temporal variations in activities.



l

Figure 3-7: Comparing different models with different time stretching factors on the RLBench dataset.

Toyota Smarthome Toyota Smarthome [Das *et al.*, 2019] is a dataset that contains videos of humans performing everyday activities such as "walk", "take pills", and "use laptop". It also comes with 3D-skeleton detections. There are around 16.1k videos in the dataset, and 19 activity classes. The videos' length varies between a few seconds to 3 minutes. We subsample 30 frames for each video. We split frames into training (9.9k), validation (2.5k), and testing (3.6k). We treat human joints as entities. The input is then the position of joints, the velocity of joints, limb lengths, and joint angles. We evaluated our model and STGCN on a 19-way classification task. We also test model performance on time-warped sequences by accelerating the trajectories by two times.

Our model achieves a comparable accuracy to STGCN on the standard classification task: (42.0% vs. 43.0%). Importantly, on the generalization test to time-warped sequences, our model has only a 0.8% performance drop (41.2%), while STGCN drops 10.7% (32.3%). This indicates that the temporal structures learned by TOQ-Net improve model generalization to varying time courses.

Volleyball Activity The volleyball dataset [Ibrahim *et al.*, 2016] contains 4830 video clips collected from 55 youtube volleyball videos. They are labeled with 8 group activities (e.g. "left spike" and "right pass"). Each video contains 20 frames with the labeled group activity performed at the 10-th frame. The dataset also includes annotations for players, including the bounding box, the indicator of whether the player is involved in the group activity, and the individual action such as "setting", "digging", and "spiking". We use the manual annotations (processed by an MLP) as the input features. We train models to classify the video into one of the eight group activities, following the original split, i.e., 24, 15, and 16 of 55 videos are used for

training, validation, and testing.

On the standard classification task, TOQ-Net achieves a comparable performance with STGCN (73.3% vs. 73.6%). When we perform time warping on the input sequences, STGCN's performance drops by more than 25.0% (39.5% on temporally shifted trajectories and 48.6% on 2× quick motion trajectories), while our model drops only 3% (70.3% on temporally shifted trajectories and 70.7% on quick motion trajectories). This again shows the generalization ability of TOQ-Net w.r.t. varying time courses, and the robustness of learned temporal structures.

3.4 Related Work

Action concept representations and learning. First-order and linear temporal logics [LTL; Pnueli, 1977] have been used for analyzing sporting events [Intille and Bobick, 1999, 2001] and activities of daily living [Tran and Davis, 2008; Brendel *et al.*, 2011] in logic-based reasoning frameworks. However, these frameworks require extra knowledge to annotate relationships between low-level, primitive actions and complex ones, or performing search in a large combinatorial space for candidate temporal logic rules [Penning *et al.*, 2011; Lamb *et al.*, 2007]. By contrast, TOQ-Nets enable end-to-end learning of complex action descriptions from sensory input with only high-level action-class labels.

Temporal and relational reasoning. This paper is also related to work on using data-driven models for modeling relational and temporal structure, such as LTL [Neider and Gavran, 2018; Camacho *et al.*, 2018; Chou *et al.*, 2020], Logical Neural Networks [Riegel *et al.*, 2020], ADL description languages [Intille and Bobick, 1999], and hidden Markov models [Tang *et al.*, 2012]. These models need human-annotated action descriptions and symbolic state variables (e.g., *pick up* x means a state transition from not *holding* x to *holding* x), and dedicated inference algorithms such as graph structure learning. In contrast, TOQ-Nets have an end-to-end design, and can be integrated with other neural networks. People have also used structural representations to model object-centric temporal concepts with graph neural networks [Yan *et al.*, 2018, GNNs;], recurrent neural networks [RNNs; Ibrahim *et al.*, 2016], and integrated GNN-RNN architectures [Deng *et al.*, 2016; Qi *et al.*, 2018]. TOQ-Nets use a similar relational representation, but different models for temporal structures.

3.5 Conclusion and Discussion

The design of TOQ-Nets suggests multiple research directions. For example, the generalization of the acquired action concepts to novel object kinds, such as from *opening fridges* to *opening bottles*, needs further exploration. Meanwhile, TOQ-Nets are based on physical properties, e.g. 6D poses. Incorporating representations of mental variables such as goals, intentions, and beliefs can aid in action and event recognition [Baker *et al.*, 2017; Zacks *et al.*, 2001; Vallacher and Wegner, 1987].

In summary, we have presented TOQ-Nets, a neuro-symbolic architecture for learning to describe complex state sequences with quantification over both entities and time. TOQ-Nets use tensors to represent the time-varying properties and relations of different entities, and use tensor pooling operations over different dimensions to realize temporal and object quantifiers. TOQ-Nets generalize well to scenarios with varying numbers of entities and time courses.

Chapter 4

Learning to Plan with Skills from Rational Demonstrations

This chapter presents RatSkills, compositional hierarchical rational skill models that support efficient planning and inverse planning for achieving novel goals and recognizing activities. We learn RatSkills by observing expert demonstrations and reading abstract language descriptions of the corresponding task (e.g., *collect* wood, *craft* a boat, and finally, *go across* the river). The learned goal-centric representation enables planning for novel objectives given in the form of either temporal task descriptions or black-box goal tests. It can also be used to infer another agent's intended task from their actions via Bayesian inverse planning. We demonstrate through experiments in both discrete and continuous domains that our learning algorithms recover a set of RatSkills by observing and explaining other agents' movements, and plan efficiently for novel goals by composing learned skills.

The material of this chapter has been previously published in Luo *et al.* [2021]. Zhezheng Luo, in particular, contributed significantly to the materials presented in this chapter.

4.1 Introduction

We study the problem of learning a compositional and planning-compatible representation of skills from demonstrations. Consider the example shown in Fig. 4-1: given a sequence of low-level state-action pairs, and a corresponding abstract task description, *collect wood <u>then</u> craft a boat <u>then</u> go across river*, humans can effortlessly decompose the observed trajectory into fragments, each of which corresponds to a particular skill named in the description. Furthermore, we learn an abstract description of individual

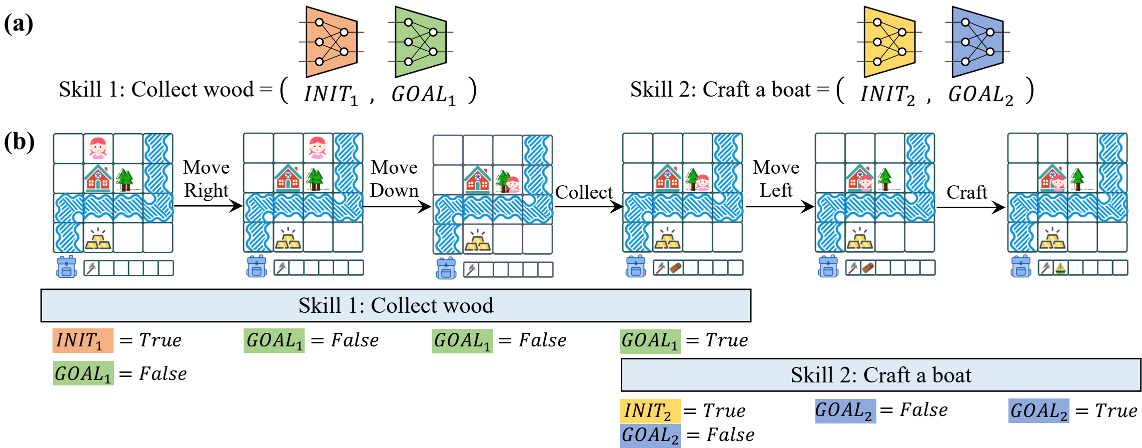


Figure 4-1: Interpreting a demonstration and its description in terms of RatSkills: (a) Each RatSkill consists of two conditions I_o and G_o. (b) The system infers a transition to the next skill if both the G condition of the current skill and the I condition of the next skill are satisfied. Such transition rules can be used to interpret demonstrations and to plan for tasks that require multiple skills to achieve.

skills, e.g., what's the goal of collecting wood, so that these skills can be flexibly recomposed in novel situations, for example, in this domain, to craft a house.

Our goal is to build machines that have this same ability. In this paper, we present RatSkills, compositional goal-centric skill models that support planning and inverse planning. RatSkills describe each skill o (e.g., *collecting wood*), with two neural network classifiers: the initial condition I_o and the goal condition G_o, denoting the conditions that are satisfied before and after the execution of the skill, respectively. Intuitively, I_o and G_o jointly characterize the important state transition that happened during the skill execution, for example, from not having wood in the agent's inventory to having wood. We say a state sequence $\bar{s} = (s_0, s_1, \cdots s_T)$ satisfies a skill o if the first state s_0 satisfies I_o (i.e., $I_o(s_0) = \mathbf{T}$ and the last state s_T satisfies G_o. In contrast to widely-adopted policy-based skill representations, our goal-centric representation allows us to compose these representations to perform novel tasks, to make plans for novel goals, and to infer other agents' intended tasks via inverse planning.

We propose to learn RatSkills by observing expert demonstrations and reading abstract language descriptions of the corresponding task (e.g., *collect* wood, *craft* a boat, and finally, *go across* the river). There are two challenges in learning from such data: first, segmenting trajectories to determine correspondences between individual skill symbols in the task description and fragments of the trajectory, and, second, learning a representation for individual skills, which can then be flexibly recomposed in planning. In general, there are many ways to segment a demonstration, both for skill

learning and task recognition, making these problems ill-posed as there are generally an infinite number of objectives that are consistent with any demonstration [Ng *et al.*, 2000].

RatSkills offer a solution to both of these challenges simultaneously, by assuming that the experts are *rational*: RatSkills generate trajectories to accomplish their target task while approximately minimizing their total cost. In RatSkills, the skill representation I_o and G_o are learned based on a Bayesian inverse planning mechanism. Given the initial state and the task description, we first use a built-in planner to obtain *rational* trajectories for completing the task. Next, we compare these trajectories with the observed demonstration and quantify the rationality score of each demonstration. Finally, we update the weights of the classifiers, using gradient descent to maximize the rationality score.

We evaluate RatSkills on two benchmarks: CraftWorld [Chen *et al.*, 2021], a grid-world domain with a rich set of object crafting tasks, and Playroom [Konidaris *et al.*, 2018], a 2D continuous domain with geometric constraints. Our model achieves comparable results with end-to-end and modular policy-based approaches on the plan recognition task and significantly outperforms them on planning tasks where the agent needs to generate trajectories itself to accomplish a given task. RatSkills show particularly superior performance in terms of compositional generalization.

4.2 Related Work

Learning from demonstration. The idea of learning from demonstration (LfD) generally refers to building agents that can interact with the environment by observing demonstrations by other experts, usually in the form of state-action sequences. Techniques for LfD can be roughly categorized into three groups: policy function learning [Chernova and Veloso, 2007; Torabi *et al.*, 2018], cost and reward function learning [Markus Wulfmeier and Posner, 2015; Ziebart *et al.*, 2008], and learning high-level plans [Ekvall and Kragic, 2008; Konidaris *et al.*, 2012]. We refer to Argall *et al.* [2009] and Ravichandar *et al.* [2020] as comprehensive surveys. In this paper, we propose to learn compositional skill models that support planning, and compare with methods that directly learns policy functions and cost functions. In contrast to approaches for learning high-level plans, which mostly focus on directly learning primitive action sequences, our representation, RatSkills are more general skill descriptions that are used by planners to generates primitive action sequences. Moreover, unlike them, RatSkills do not use similarities between skills [Niekum *et al.*, 2012] for segmenting the observed state-action sequences. In contrast, we segment the

demonstration and associate skill labels for each fragment by assuming the expert is *rational*.

Modular skill models. There have been a significant number of recent approaches that use deep neural networks to construct modular skills models for interaction. Researchers have proposed models for learning these models by simultaneously looking at the state-action sequence and reading task specifications in the form of skill sequences [Corona *et al.*, 2021; Andreas *et al.*, 2017; Andreas and Klein, 2015], programs [Sun *et al.*, 2020], and linear temporal logic (LTL) formulas [Bradley *et al.*, 2021; Sadigh *et al.*, 2014; Toro Icarte *et al.*, 2018; Tellex *et al.*, 2011]. However, they either require additional annotation for the segmenting the sequence and associating fragments with labels in the task description [Corona *et al.*, 2021; Sun *et al.*, 2020], or cannot learn models for planning and execution from demonstration [Tellex *et al.*, 2011]. By contrast, in RatSkills, we use a small but expressive subset of LTL sentences for describing tasks and propose to jointly learn skill models and segment the demonstration sequence based on the *rationality* assumption.

Inverse planning. Our model is also related to methods for inferring agents' intentions by observing their states and actions, by assuming agents are *rationally* selecting actions to achieve their goal. The technique of *inverse planning* addresses this problem by finding a task description t that maximizes the consistency between the agent's behavior and the synthesized plan for t [Baker *et al.*, 2009]. While existing work has largely focused on modeling the rationality of agents [Baker *et al.*, 2009; Zhi-Xuan *et al.*, 2020] and more expressive task description languages [Shah *et al.*, 2018], it generally does not jointly consider how models for skills can be learned so that the agent itself can generate plans to achieve novel tasks. In RatSkills, we use the Bayesian inverse planning framework [Baker *et al.*, 2009] for inferring agents' intentions and present learning algorithms that can learn *rational* skill models from demonstration.

4.3 Planning and Learning of RatSkills

4.3.1 Problem Formulation

We assume an environment in the form of a deterministic process with states, actions, transition function and cost function $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$. In this formulation, we assume that $\mathcal{S}$ is given in an *object centric* representation, in which each state is specified in terms of the values of a set of properties and relations applied to a universe of objects;

the properties and relations are fixed for any given domain, but the universe may change for different problem instances and our model will generalize over problems with different numbers of objects. Other state representations, such as images, would be consistent with this overall framework. The primitive action space $\mathcal{A}$ must be such that it can be used by a low-level planner to find state trajectories through the space: we illustrate both a discrete $\mathcal{A}$ and a continuous robot-motion $\mathcal{A}$. We assume that the agent has access to the transition and cost models, although they could in principle be learned prior to or in parallel with the process we describe. We will say that a state sequence $\bar{s} = (s_1, \ldots, s_n) \in \mathcal{S}^n$ is *feasible*, if for all $i \in (1, \ldots, n-1)$, $\exists a \in \mathcal{A}. s_{i+1} = \mathcal{T}(s_i, a)$.

Our objective is to learn a set of *rational skills* (RatSkills), where each RatSkill has an atomic skill name o, and is specified in terms of a pair of classifiers (I_o, G_o), each of which maps $\mathcal{S}$ into Boolean values. In our formulation, we use *neural logic machines* (NLMs) [Dong *et al.*, 2019] to represent these classifiers, because they provide flexible representation of first-order logic formulas with finite quantification, allowing RatSkills to generalize effectively to domains with different universes of objects. We say a RatSkill is *valid* if

$$\forall s \in \mathcal{S}.\ I_o(s) \rightarrow \exists \bar{a} \in \bar{\mathcal{A}}.\ G_o(\mathcal{T}(s, \bar{a})) \ \ ,$$

where $\bar{\mathcal{A}}$ is the set of finite sequences of actions; that is, if I_o is true of some state s, then there is another state that is reachable from s in which G_o is true. Unlike the skills often defined in terms of unconstrained policies, our skills are *rational* in the sense that it still requires reasoning to execute them. To execute RatSkill o from some state $s \in I_o$, we call a planner with G_o as the goal condition; as long as o is valid, we will obtain a plan, which is then executed to reach some state $s' \in G_o$. Although this *model-based* skill representation pays a computational cost at execution time, in domains and skills that we are interested in, achieving G_o usually requires only a small number of steps. More importantly, it exhibits generalization abilities that far exceed those of stored policies.

We hope to use these skills to improve planning efficiency and to enable execution of high-level instructions as well as to recognize high-level intentions of other rational agents. To do so, we define a simple temporal language $\mathcal{TL}$ for composing skills. Syntactically: all primitive skills o are in $\mathcal{TL}$; and for all $t_1, t_2 \in \mathcal{TL}$, $(t_1 \textbf{ and } t_2)$, $(t_1 \textbf{ or } t_2)$, and $(t_1 \textbf{ then } t_2)$ are all in $\mathcal{TL}$. Semantically, a feasible state sequence $\bar{s}$ satisfies a task description t, written $\bar{s} \models t$ in the following cases:

- If t is a *RatSkill* o, then $I_o(s_1)$ and $G_o(s_n)$.
- If $t = (t_1 \textbf{ or } t_2)$ then $\bar{s} \models t_1$ or $\bar{s} \models t_2$.

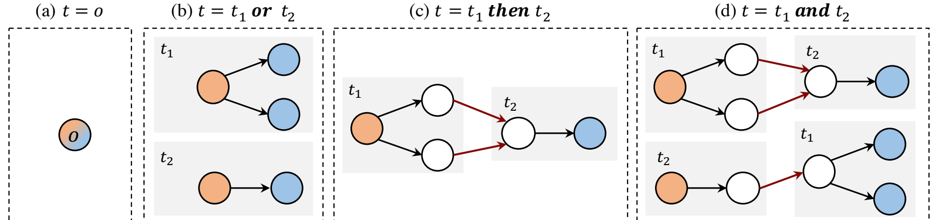


Figure 4-2: Illustrative example of how finite state machines (FSM) are constructed for tasks.

- If $t = (t_1 \textbf{ then } t_2)$ then $\exists 0 < j < n$ such that $(s_1, \ldots, s_j) \models t_1$ and $(s_j, \ldots, s_n) \models t_2$.
- If $t = (t_1 \textbf{ or } t_2)$ then $\bar{s} \models (t_1 \textbf{ then } t_2)$ or $\bar{s} \models (t_2 \textbf{ then } t_1)$.

The language $\mathcal{TL}$ can be viewed as a small set of linear temporal logic (LTL) sentences, with only propositional connectives, as well as the *then* and the *eventually* quantifiers. This simple grammar covers all instructions that we are considering in this paper but could be extended if necessary. In the following sections, we describe our strategies for using and learning RatSkills.

4.3.2 Task-Augmented Transition Models

We will frequently want to construct plans for, or evaluate plans relative to, a particular task $t \in \mathcal{TL}$. To do so, we will construct a new deterministic transition system called a *task-augmented transition model*. This construction is a variation on those used to bias reinforcement-learning [Parr and Russell, 1998] and to plan to meet LTL specifications [Belta, 2016].

We begin by constructing a finite state machine FSM_t based on the task specification t. Each FSM_t is a tuple $\langle V_t, E_t, VI_t, VG_t \rangle$, where V_t is the set of skill nodes in FSM_t, and E_t the edges. An edge $(x, y) \in E_t$ indicates that after executing skill x, the agent can switch to the execution of another skill y. All possible starting skill nodes of the state machine are indicated by $VI_t \subseteq V_t$. Analogously, the set of terminal nodes of the machine is $VG_t \subseteq V_t$.

The finite state machines can be constructed based on the recursive structure of $\mathcal{TL}$. Fig. 4-2a-d shows example constructions for the four cases.

- (a) If t is a *RatSkill* o, then $\text{FSM}_t \coloneqq \langle \{o\}, \emptyset, \{o\}, \{o\} \rangle$, which is a simple FSM containing only one node, which corresponds to the skill.
- (b) If $t = (t_1 \textbf{ or } t_2)$ then $\text{FSM}_t \coloneqq \langle V_{t_1} \cup V_{t_2}, E_{t_1} \cup E_{t_2}, VI_{t_1} \cup VI_{t_2}, VG_{t_1} \cup VG_{t_2} \rangle$. Essentially, we construct a new FSM that composes t_1 and t_2 in parallel. Executing either FSM_{t_1} or FSM_{t_2} will complete the task.
- (c) If $t = (t_1 \textbf{ then } t_2)$ then $\text{FSM}_t \coloneqq \langle V_{t_1} \cup V_{t_2}, E_{t_1} \cup E_{t_2} \cup K(VG_{t_1}, VI_{t_2}), VI_{t_1}, VG_{t_2} \rangle$.

A set of new edges $K(VG_{t_1}, VI_{t_2})$ connects all pairs (x, y) where $x \in VG_{t_1}$ and $y \in VI_{t_2}$.

- (d) If $t = (t_1$ **and** $t_2)$, we rewrite t as $(t_1$ **then** $t_2)$ **or** $(t_2$ **then** $t_1)$ and construct FSM_t.

For simplicity, we will add two special nodes: the super-starting node v_0, which connects to all starting nodes VI_t and the super-terminal node v_T, to which all terminal nodes VG_t are connected. The initial and goal conditions for both super nodes evaluate to true for all states in $\mathcal{S}$.

We now formally define our *task-augmented transition model* for a given task d, $TATM_d = \langle \mathcal{S}', \mathcal{A}', \mathcal{T}', \mathcal{C}' \rangle$, by composing the FSM with the basic environmental model. Concretely, $\mathcal{S}' = \mathcal{S} \times V_t$. We denote each task-augmented state as (s, v), where s is the environment state of the agent, and v indicates the skill being currently executed. The actions $\mathcal{A}' = \mathcal{A} \cup E_t$, where each action either corresponds to a primitive action $a \in \mathcal{A}$ or a transition in FSM_t, such as finishing executing the current skill and proceeding to the next skill. We further define $\mathcal{T}'((s, v), a) = (\mathcal{T}(s, a), v)$ if a is a primitive action in $\mathcal{A}$, while $\mathcal{T}'((s, v), a) = (s, v')$ if $a = (v, v') \in E_t$ is an edge in the FSM. Similarly, for the cost function,

$$\mathcal{C}'((s, v), a) = \begin{cases} \mathcal{C}(s, a) & \text{if } a \in \mathcal{A}, \\ -\lambda \left(\log G_v(s) + \log I_{v'}(s)\right) & \text{if } a = (v, v') \in \mathbb{E}_t, \end{cases}$$

where λ is a hyperparameter. The key intuition behind the construction of $\mathcal{C}'$ is that the cumulative cost of any trajectory from v_0 to v_T is the summation of all primitive action costs added to the log-likelihood of all the skill sub-sequences *satisfying* their associated subtask description —that is, at each skill transition, the world state s should satisfy both the goal condition of the current skill and the initial condition of the next skill.

4.3.3 Planning and Inverse Planning with RatSkills

We can use a set of RatSkills in three ways: planning to achieve a goal, planning to follow a sequence of instructions, and *inverse* planning to infer another agent's intended task from an action sequence. A critical basic component is planning to execute a single RatSkill o. In the case of discrete $\mathcal{A}$, this reduces to a graph-search problem with goal-test G_o; we use A^* with a learned value function as a heuristic, as described below. In the case where $\mathcal{A}$ corresponds to robot motion planning, we use the RRT [LaValle and others, 1998] algorithm.

Planning for a task described in $\mathcal{TL}$. RatSkills directly support making plans for completing a task t specified with $\mathcal{TL}$. Concretely, our goal is to find an optimal-cost sequence of actions in the task-augmented transition model from (s_0, v_0) to (s_T, v_T), where s_0 is the initial state of the environment, and s_T is an arbitrary state in $\mathcal{S}$, which is the last state reached by the agent when it completes the last skill in t. We make plans using slightly modified versions of A^* search for discrete domains and Rapidly-exploring Random Tree (RRT) for continuous domains. Both of these algorithms can be viewed as doing forward search to construct a trajectory from a given state to a state that satisfies the goal condition. Extensions are required to handle the hierarchical task structure of the FSM at the high level and the primitive actions at the low level.

Our modified A^* search maintains a priority queue of nodes to be expanded. Each node is associated with a heuristic value which adds up the total cost of the agent reaching this state and an estimated cost-to-go, in our case, either a uniform heuristic for novel tasks, or a heuristic produced by a learned value function approximator for previously-seen tasks. At each step, instead of always popping the task-augmented state (s, v) with the optimal heuristic value, we first sample a skill node v in the FSM, and then choose the priority-queue node with the smallest heuristic value among all states $(\cdot, v)$. This search algorithm remains complete, but balances the time allocated to finding a successful trajectory for each RatSkill in the FSM. Similarly, for RRT, we maintain different RRTs for different skill nodes in the FSM and balance the number of nodes we expand for each tree. Implementation details of these modified versions of A^* and RRT are provided in the supplementary material.

Planning for a goal state without a task description. RatSkills also support efficient planning for novel goals without a task description in $\mathcal{TL}$. In this paper, we study the case where a black-box goal-state test G^* is provided to the algorithm. The task is formulated as finding a sequence of actions that leads to a state s where $G^*(s)$. We use RatSkills for this task with a hierarchical search mechanism. First, we enumerate candidate skill sequences, i.e., tasks in $\mathcal{TL}$ composed with only skills and the *then* connectives. Next, we run parallel A^* search procedures for each candidate high-level skill plan. The algorithm will terminate and return a plan when any one of the downward refinements reaches a state that satisfies G^*. We leave the integration of other informed search methods into the high-level skill space as future work.

Bayesian inverse planning. The set of RatSkills can also be used to infer another agent's intended task from their action sequence via Bayesian inverse planning. Con-

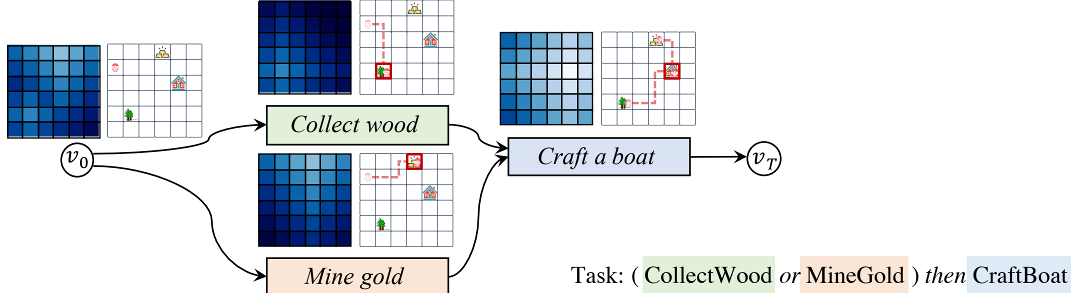


Figure 4-3: An example of the value function $J_t(s, v, a)$ for task-augmented states on a simple FSM. $\max_{a \in \mathcal{A}} J_t(s, v, a)$ are plotted at each location at each FSM node. Deeper color indicates larger cost. Dotted lines illustrate one *rational* trajectory for each skill; red boxes indicate goals.

cretely, the input to our algorithm is a state sequence $\bar{s}$ and an action sequence $\bar{a}$, such that $\forall i, s_{i+1} = \mathcal{T}(s_i, a_i)$. It is important to note that the state-action sequence only contains the environment states $\mathcal{S}$ and actions $\mathcal{A}$. However, we do not know the FSM state $v_i \in V_t$ associated with each state s_i, nor the FSM transition actions $e \in E_t$ (a.k.a. the skill *segmentation* of the state-action sequence). In addition to the state and action sequences, we provide the algorithm with a set of candidate tasks described in $\mathcal{TL}$. The output of the algorithm is a ranking of all candidate tasks, by how likely they are the intended task of the agent.

For each candidate task t, our inverse planning procedure starts by running forward search (A^* or RRT) from a seed node set composed of all task-augmented states (s, v) based on the task-augmented transition model of t, where $s \in \bar{s}$ and $v \in V_t$. This step produces a larger set of task-augmented states $\mathcal{K}_t$. We run Bellman-Ford (analogous to value iteration for deterministic systems) on $\mathcal{K}_t$ and obtains a value function $J_t(s, v, a)$ (analog to the Q-value for MDPs but in terms of cost) for all nodes along the observed state sequence $s \in \bar{s}$, all FSM states $v \in V_t$, and all actions $a \in \bar{a}$. Here, $\bar{s}$ are environment states $s_i \in \mathcal{S}$, and $\bar{a} = \{a_i\}$ contains both environmental actions $\mathcal{A}$ and skill transition actions E_t. Thus, this task-augmented action sequence $\bar{a}$ uniquely yields a FSM state sequence $\bar{v} = \{v_i\}$. Fig. 4-3 shows an example of the value function over the task-augmented states.

Based on the J-value, we define the *rationality* of a state-action sequence relative to a task t as,

$$\mathrm{Rat}\left(\bar{s}, \bar{a}, t\right) \coloneqq \prod_{i=0}^{|\bar{s}|-1} \mathrm{Rat}\left(s_i, v_i, a_i, t\right), \quad \mathrm{Rat}\left(s_i, v_i, a_i, t\right) \coloneqq \frac{\exp\left(-\alpha \cdot J_t(s_i, v_i, a_i)\right)}{\int_{a' \in \mathcal{A} \cup E_t} \exp\left(-\alpha \cdot J_t(s_i, v_i, a')\right)},$$

where α is a scalar hyperparameter. For discrete action spaces, the integral is simply a finite sum. In continuous action spaces, we use Monte Carlo sampling to compute

the integral. The definition of rationality allows us to define a score function

$$\mathit{score}(\bar{s}, \bar{a}, t) \coloneqq \sum_{e_i=(v,v')\in\bar{a}\cap E_t} (\log G_v(s_i) + \log I_{v'}(s_i)) + \log \mathrm{Rat}\,(\bar{s}, \bar{a}, t)\,,$$

which evaluates both the rationality of the agent's actions and the validity of their skill transitions. Next, we employ a dynamic programming (DP) based approach to find a set of skill transition points along the observed state-action sequence $\bar{s}$ and $\bar{a}$, (i.e., to insert a number of skill transition actions $e \in E_t$ and max over the latent FSM states). Formally, let $\mathit{score}'(v, \bar{s}_{:n}, \bar{a}_{:n-1}, t)$ be the maximum score of the first n steps of the state-action sequence, when the agent is executing the skill v. There are two possible DP transitions we can make: a_n and $e = (v, v') \in E_t$. Taking a_n yields a candidate score for $\mathit{score}'(v, \bar{s}_{:n+1}, \bar{a}_{:n}, t) = \mathit{score}'(v, \bar{s}_{:n}, \bar{a}_{:n-1}, t) + \mathrm{Rat}(s_n, v, a_n, t)$. Taking e yields a candidate score for $\mathit{score}'(v', \bar{s}_{:n}, \bar{a}_{:t-1}, t) = \mathit{score}'(v, \bar{s}_{:n}, \bar{a}_{:n-1}, t) + \mathrm{Rat}(s_n, v, e, t) + \log G_v(s_n) + \log I_{v'}(s_n)$. Solving this dynamic program requires $O(|\mathcal{S}| \times |V_t|)$ time and space. We define $\mathit{score}(\bar{s}, \bar{a}, t)$ as $\mathit{score}'(v_T, \bar{s}, \bar{a}, t)$, essentially the maximum possible score that we can get by inserting skill transition actions into $\bar{a}$. Finally, we compute the likelihood of the agent's intention as:

$$p(t|\bar{s}, \bar{a}) = \frac{\exp\left(\beta \cdot \mathit{score}(\bar{s}, \bar{a}, t)\right)}{\sum_{t'} \exp\left(\beta \cdot \mathit{score}(\bar{s}, \bar{a}, t')\right)},$$

where t' enumerates over all candidate tasks, and $\beta = 1$ is a scalar hyperparameter.

4.3.4 Learning RatSkills

We employ a contrastive-learning based approach to learn a set of RatSkills from paired task descriptions $t \in \mathcal{TL}$ and state-action sequences $(\bar{s}, \bar{a})$, which we assume are generated by a *rational* agent to accomplish t. The intuition is that the model parameters should be adjusted so that each task description in the training set is a high-probability rational explanation of the accompanying state-action sequence. In order to learn the weights in the classifiers I_o and G_o, we define the following training objective:

$$\mathcal{L} = \sum_{(\bar{s},\bar{a},t)\in\mathcal{D}} \mathit{score}(\bar{s}, \bar{a}, t) + \gamma \cdot \log \frac{\exp\left(\beta \cdot \mathit{score}(\bar{s}, \bar{a}, t)\right)}{\sum_{t'} \exp\left(\beta \cdot \mathit{score}(\bar{s}, \bar{a}, t')\right)},$$

where $\mathcal{D}$ is the dataset we train on, and t' are randomly sampled negative tasks in $\mathcal{TL}$ for the particular data point. Since the cost function for task-augmented transition models $\mathcal{C}'$ (and thus the computed J function) is fully differentiable w.r.t. I_o and

G_o for all RatSkills o, we can simply use gradient descent to maximize $\mathcal{L}$. From a Bayesian inverse planning perspective, we are running back-propagation through the inverse planning. Finally, we should note that the computation of $\mathcal{L}$ only requires J values for states along the observed state-action sequences, instead of for the entire state-action space, making our model scalable to compositional state spaces with many objects and obstacles.

4.4 Experiments

We compare our model with other skill-learning approaches in two environments: Crafting World [Chen *et al.*, 2021] and Playroom [Konidaris *et al.*, 2018]. In both cases, we provide expert demonstrations generated by human-written programs, and evaluate the learned skill representations on two tasks: planning and inverse planning.

4.4.1 Setup

Problems. To evaluate planning, each algorithm is given a novel task, either specified in $\mathcal{TL}$, or as a black-box goal state classifier. The objective is to generate a trajectory of actions from the agent's current state to complete the task. In the problem of inverse planning, we give each algorithm a state-action sequence, from which the algorithm should infer the intended task of the agent. We provide each algorithm with a list of candidate task descriptions in $\mathcal{TL}$ for which it outputs a ranking. For both tasks, we focus on evaluating the compositional generalization of different algorithms, i.e., how they interpret and infer complex tasks that they have not seen during training.

Baselines. We compare RatSkills against three baselines. They all learn from demonstration but learn different underlying representations: IRL methods learn reward-based representations and behavior cloning methods directly learn policies. Thus, we study how the learned representation affects generalization and overall performance. We provide the implementation details of all baselines in the supplementary material.

1. Behavior cloning [BC; Torabi *et al.*, 2018] directly learns a task-conditioned policy that maps from the current state of the agent and the given task to a primitive action in the environment. For planning, we directly follow the task-conditioned policy. For inverse planning, we rank all candidate tasks t by the consistency between the task-conditioned policy on t and the observed state-action sequence.
2. Max-Entropy inverse reinforcement learning [IRL; Ziebart *et al.*, 2008] learns a task-conditioned reward function by trying to explain the demonstration. For

planning, we use the built-in deep-Q-learning algorithm to find the optimal policy conditioned on the task description. Similar to BC, for inverse planning, we rank all candidates by the consistency between the observed sequence and the derived task-conditioned policy. For both BC and IRL approaches, we use an LSTM network [Hochreiter and Schmidhuber, 1997] to encode the task description in $\mathcal{TL}$.

3. BC-FSM is the BC algorithm augmented with our FSM description of tasks. Specifically, in contrast to learning the initial and goal conditions for each individual skill, BC-FSM learns a policy and a termination condition classifier for each skill. Thus, instead of segmenting the demonstration sequence based on the rationality assumption, BC-FSM segments them based on how consistent each fragment is with the policy for the corresponding skill. We use the same hierarchical search and dynamic programming algorithm as RatSkills for BC-FSM. That is, in planning, we perform balanced policy rollouts for all possible branches in an FSM, and select the action sequence with the smallest cost. In inverse planning, we max over all possible latent FSM states.

For the inverse planning task, we include an additional sequence classification baseline: LSTM. It uses two separate LSTM networks to encode the state-action sequence and the task description, respectively. Next, it uses a multi-layer perceptron (MLP) to compute a scalar score for each candidate task description. We use a Softmax-CrossEntropy loss to train this model.

4.4.2 Environments

Crafting World. Our first environment is Crafting World [Chen *et al.*, 2021], a Minecraft-inspired crafting environment. The agent can move in a 2D grid world and interact with objects next to it, including picking up tools, mining resources, and crafting items. Mining in the environment typically requires tools, while crafting tools and other objects has preconditions, such as being close to a workstation or holding another specific tool. Thus, crafting a single item often takes multiple steps. In certain maps, there are also obstacles such as rivers (which requires boats to go across) and doors (which requires specific keys to open).

We define 26 primitive tasks, instantiated from templates of *grab X*, *toggle switch*, *mine X*, and *craft X*. While generating trajectories, we make sure that all required items have been placed in the agent's inventory. For example, before mining wood, the agent already has an axe in their inventory. In this case, the agent is expected to move to a tree and execute the mining action. We also define another 26 compositional tasks that are composed of the aforementioned primitive tasks.

Figure 4-4: An illustration of the Playroom environment and a trajectory for the task: *turn on the music then play the ball then turn off the music*.

We train all models on these primitive and compositional tasks and test them on two splits: *compositional* and *novel*. The *compositional* split contains novel state-action sequences of previously-seen tasks. The novel split contains rational state-action sequences for 12 novel tasks that are composed of the primitive tasks, but have never been seen during training.

Playroom. Our second environment is Playroom [Konidaris *et al.*, 2018], a 2D maze with continuous coordinates and geometric constraints. An illustrative example of the environment and a trajectory are shown in Fig. 4-4. Specifically, a 2D robot can make moves in a small room with obstacles. The agent has three degrees-of-freedom (DoFs): x and y direction movement, and a 1D rotation. The environment invalidate movements that cause collisions between the agent and the obstacles. Additionally, there are six objects initially randomly placed in the room, which the robot can interact with. For simplicity, when the agent is close to a certain object, the corresponding robot-object interaction will be automatically triggered.

Similar to the Crafting World, we have defined six primitive tasks (corresponding to the interaction with six objects in the environment) and eight compositional tasks (e.g., *turn on the music then play with the ball*). Similarly, we made up another eight novel tasks. We will train different models on rational demonstrations for both the primitive and compositional tasks, and evaluate them on the compositional and novel splits.

4.4.3 Results

Planning for a task. Table 4.1 summarizes the results for planning in both environments. RatSkills outperforms all baselines. On the *compositional* split, our model

Model	Crafting World		Playroom	
	Com.	Novel	Com.	Novel
IRL	36.5	1.8	28.3	9.6
BC	11.2	0.8	15.8	4.8
BC-FSM (ours)	5.2	0.3	38.2	31.5
RatSkills (ours)	**99.6**	**97.8**	**82.0**	**78.2**

Table 4.1: Results of the planning task for the Crafting World environment and the Playroom environment. All models are trained on two data splits: *primitive* and *compositional*. They are evaluated on the *compositional* and the *novel* split, which contains unseen task descriptions.

achieves nearly perfect success rate in the Crafting World (99.6%) and high performance in Playroom (82.0). Comparatively, although the tasks have been presented during training of all baselines, their scores remain below 40%.

On the *novel* split, RatSkills outperforms all baselines by a larger margin than on the *compositional* split, in both environments. We observe that, in Crafting World, since some *novel* tasks contain longer descriptions than those in the *compositional* set, all baselines have almost zero success rate on them.

In Fig. 4-5, we further investigate the performance of RatSkills on each *novel* task in the Crafting World environment. RatSkills achieve at least 85% success rate for all novel tasks, reaching 100% success rate for the novel tasks composed of 6 or fewer skills. In general, the success rate drops when the task becomes more complex. Note that the most complex task, *craft arrow*, contains 14 primitive skills.

In particular, we have also found our RatSkills generalize well to environments with movement constraints (doors and rivers). For example, agents can accomplish all tasks in the *compositional* split without crafting a boat, whereas in tasks represented with blue and orange dots, crafting boats is a prerequisite for their movements.

Planning with a black-box goal test. We also evaluate RatSkills on planning with a black-box goal test. These problems require a long solution sequence, making them too difficult to solve with blind search from an initial state. Since there is no task specification given, in order to solve the problems efficiently, it is critical to use RatSkills for search guidance.

We manually designed three goal tests. They require sequential executions of 2, 3, and 4 primitive skills to complete, respectively. We run our hierarchical search algorithm based on RatSkills and a blind forward-search algorithm on 100 random initial states for all three tasks. Fig. 4-6 summarizes the result. Overall, RatSkills enable efficient search for plans. On relatively easier tasks (2 or 3 steps), search with

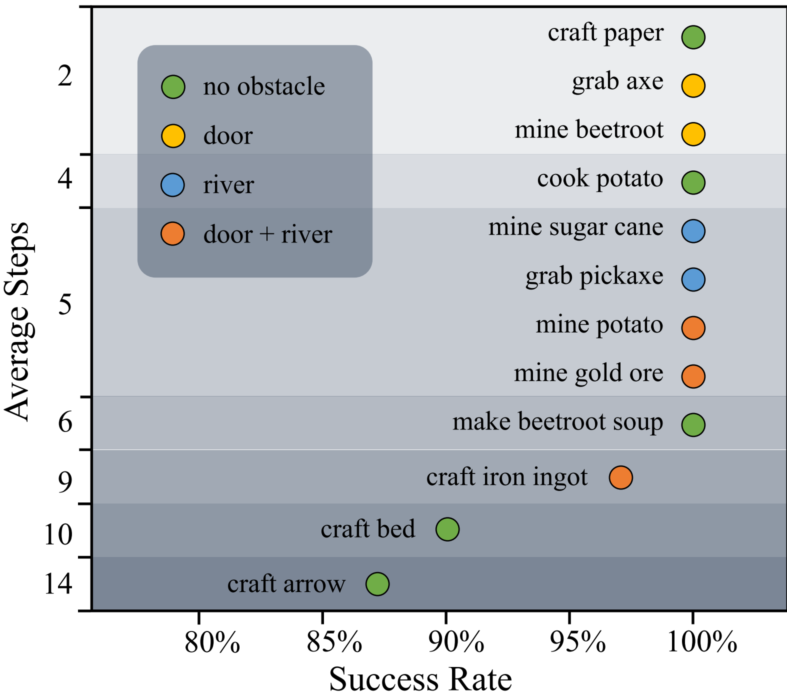


Figure 4-5: Planning success rate on 12 novel tasks in the Crafting World environments, sorted by the number of skills in the task description. We use different colors to represent environments with different obstacles. Green: no obstacles. Yellow: there are doors to be opened by specific keys. Blue: the agent must craft a boat to go across rivers. Orange: the environment has both doors and rivers.

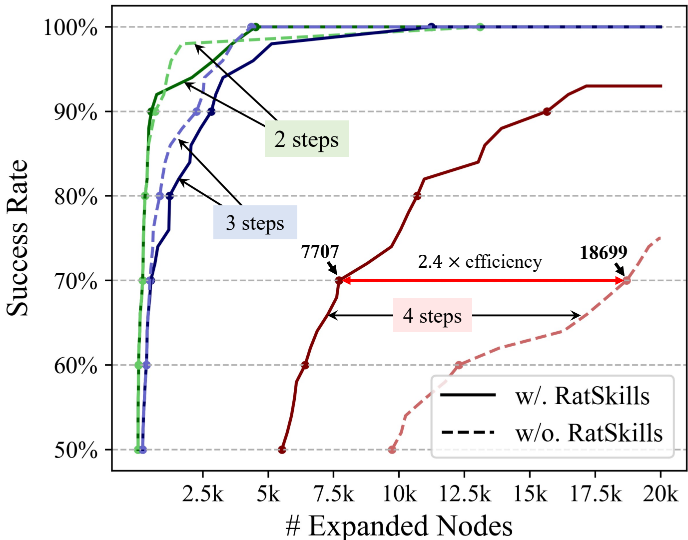


Figure 4-6: RatSkills can be integrated into a simple forward-search algorithm to improve the success rate w.r.t. the same number of expanded nodes in the search tree, because the learned skill models suggest meaningful subgoal states. We do evaluation on 3 planning tasks in the Crafting World environment. We use 100 random initial states for each task.

	Crafting World		Playroom	
Model	Com.	Novel	Com.	Novel
LSTM	**100.0**	25.0	51.1	37.5
IRL	64.1	22.1	92.5	65.6
BC	85.6	20.8	72.5	45.0
BC-FSM (ours)	**100.0**	**100.0**	85.0	**98.8**
RatSkills (ours)	97.2	95.7	**97.5**	91.9

Table 4.2: Results of the inverse planning task for the Crafting World environment and the Playroom environment. The goal is to predict the most probable intended task of the agent by observing their state-action sequences. All models are trained on the union of the *primitive* and *compositional* data splits and evaluated on the *compositional* and the *novel* split.

and without RatSkills have similar efficiency. However, when the task becomes more complex (4 steps), our model significantly improves the efficiency. For example, in order to complete 70% of the problems that require 4 skills, search with RatSkills only need to expand 7707 nodes, while without RatSkills, it needs 18699 (2.4×) nodes. It is worth noting that, as the map grows larger, the effectiveness of RatSkills will become bigger, because they provide cues for meaningful sub-goals to achieve, in order to complete the final goal. We leave learning models for more efficient skill-level planning [Konidaris *et al.*, 2018] as future work.

Inverse planning. Table 4.2 shows the results for the inverse planning task. In general, models with FSM-based task representations (BC-FSM and RatSkills) outperform those that treat the task description directly as an additional input. In both environments and on both splits, RatSkills consistently perform well.

In both environments, the LSTM, IRL and BC baselines all achieve high accuracy on previously-seen tasks (i.e., in the *compositional* split) but experience a huge performance drop when generalizing to the *novel* split. By contrast, BC-FSM and RatSkills significantly outperform all other methods on the *novel* split. This suggests the effectiveness of our task-augmented transition models.

Limitations There are certain limitations of RatSkills in terms of computational cost and scalability.

First, the low-level planning part in RatSkills is done with the basic A* search (for discrete environments) and RRT (for continuous environments). These methods do not scale up well to discrete environments with many actions and continuous environments with high-dimensional states. Future work may consider incorporating more advanced planners, such as the ones using learned models to bias the search/sampling process.

Second, planning in the high-level – planning with the RatSkills– is current done with a blind search. Although we have already demonstrated its effectiveness in the planning with black-box goal test experiments, future work can further extend its applicability to longer-horizon tasks by learning causal models at the skill level [Konidaris *et al.*, 2018].

Finally, the skills learned by RatSkills are not "lifted." That is, the skills do not generalize to unseen objects (e.g., from "mine gold ore" to "mine iron." Designing new models for skill representations that generalize to unseen objects is another meaningful direction.

4.5 Conclusion

We present RatSkills, compositional hierarchical skill models that support efficienct planning and inverse planning to achieve novel goals and recognize activities of other agents. RatSkills can be learned by observing expert demonstrations and reading task specifications described in a simple task language $\mathcal{TL}$. Specifically, we present a learning algorithm based on Bayesian inverse planning to simultaneously segment the trajectory into fragments, which correspond to individual skills, and learn planning-compatible models for each skill. Our experiments in both discrete and continuous domains suggest that RatSkills have strong compositional generalization to novel goals

composed of the previously-seen skills.

In the deployment of systems based on our technique, it would be important to consider (a) consent of users supplying the state/action sequences; (b) biases that could enter via the data distribution; (c) problems of value-alignment and whether the objectives being specified by the users are ethically valid; and (d) the safety of actions that may be generated by robot following a strategy generated by this system. However, we are optimistic that researchers and developers can build upon these idea to empower useful systems for social good.

Chapter 5

Conclusion

In this thesis, we have discussed a general paradigm of integrating machine learning with structured, logic-like representations for relational and temporal events and actions. This results in a neuro-symbolic framework that augments deep neural networks with a compact but generic set of structural biases: scenes are composed of objects; events are temporally related; actions have preconditions and goals. The proposed models, TOQ-Nets and RatSkills, can be trained with gradient descent, and support compositional generalization in reasoning about events and planning for novel goals. These work suggested the following future directions.

First, as a direct extension of the RatSkills paper, we should further consider learning preconditions of skills by actively making experiments in the environment and combine skill learning and domain-independent heuristics in AI planning such as Fast-Downward [Helmert, 2006]. To elaborate, I believe the key ingredient in efficient planning is factorized state representations. For example, by discovering concepts "having wood" and "near the workstation", the agent can describe preconditions for the skill "crafting boat" with a conjunction of two concepts. We can further exploit these compositional structures to compute heuristic functions Helmert [2006] and suggest helpful actions at each step, to make the skill-level planning efficient.

Second, in both TOQ-Nets and RatSkills we have been focusing on learning, reasoning, and planning with high-level, abstract state representations, such as the positions and velocities of objects. How such frameworks can be integrated with visual recognition models to work with images and 3D representations is another meaningful direction.

Third, future work may also consider learning relational and temporal representations from language. First, language is perhaps one of the easiest ways that we can gather data from human to teach our machines. Second, language structures reflects a significant amount of structures in humans' mental representations, from which we

can get inspirations. Third, languages are natural ways to provide new information and specifications to machines.

Integrating all these research threads will enable us to build intelligent machines that are "born as a baby and learn as a child". The structures of the representations and the algorithms for reasoning and planning are built into the system. These structures form a minimal amount of prior knowledge but are generic and crucial: scenes are composed of objects; events are temporally related; actions have preconditions and goals. A human can teach robots new concepts by showing them images, talking to them, demonstrating skills, and allowing them to act in the world. After learning, queries or instructions can be provided to the robots. The robots can answer queries and make plans to accomplish the goal specified by humans.

Appendix A

Implementation Details for TOQ-Nets

This appendix chapter provides implementation details for TOQ-Nets introduced in Chapter 3.

A.1 Implementation Details

In this section, we present the implementation detail of our model, the TOQ-Nets, and five baselines (STGCN, STGCN-LSTM, STGCN-MAX, Space-Time and Non-local), including the model architecture, input features, and the training methods.

A.1.1 TOQ-Nets

In the soccer event detection task, we use three object quantification layers and three temporal quantification layers. Each object quantification layer has a hidden dimension of 16 (i.e., all nullary, unary, and binary tensors have the same hidden dimension of 16). Each temporal quantification layer has a hidden dimension of 48.

In the manipulation concept learning task, we use three object quantification layers and three temporal quantification layers. Each object quantification layer has a hidden dimension of 48. Each temporal quantification layer has a hidden dimension of 144.

Input feature extractor. We use the following physics-based input feature extractor for soccer event detection. Given the trajectories of all entities, including all players and the ball, we first compute the following physical properties: ground speed (i.e., the velocity on the xy-plane), vertical speed (i.e., the velocity along the

z direction), and acceleration. We also compute the distance between each pair of entities.

After extracting the physical properties, we use the following feature templates to generate the input features. For each physical property, X, we first normalize X across all training trajectories, and then we create $c = 5$ features:

$$\sigma\left(\frac{X - \theta_i}{\beta}\right),$$

where σ is the sigmoid function, β is a scalar hyperparameter, $\theta_i, i = 1, 2, 3, 4, 5$ are trainable parameters. These feature templates can be interpreted as differentiable implementations of $\mathbf{1}[X > \theta_i]$, where $\mathbf{1}[\cdot]$ is the indicator function. During training, we make β decay exponentially from 1 to 0.001.

In the manipulation concept learning task, we use a single fully-connected layer with sigmoid activation as the feature extractor. The hidden dimension of the layer is 64.

A.1.2 STGCN

We use the same architecture as Yan *et al.* [2018]. Table A.4 summarizes the hidden dimensions and the kernel sizes.

The STGCN model's output is a tensor of size $(T/4) \times N \times 256$, where T is the length of the input trajectory, and N is the number of entities. Following Yan *et al.* [2018], we perform an average pooling over the temporal and the entity dimension and get a feature vector of size 256. We apply a softmax layer on top of the latent feature to classify the action.

Input feature extractor. In the soccer event detection task, we do not input the distance between each pair of entities as we do for the TOQ-Nets, as the STGCN model does not support binary predicate inputs (see STGCN-2D for an ablation study). In the manipulation concept learning task, STGCN uses the same input format as the TOQ-Nets. Specifically, we represent the state of each entity with 13 unary predicates, including 3D position, 3D orientation, and bounding boxes.

A.1.3 STGCN-MAX

The STGCN-MAX model is a variation of STGCN. During graph propogation, for each node, instead of taking average of all propagated information from all neighbors, we take their maximum value over all feature dimensions, and update the hidden state

	Reg.	New	All
TOQ-Net	**87.7**	**52.2**	**79.8**
STGCN-2D	84.1	46.9	75.8

Table A.1: Ablation study of the STGCN-2D model on the soccer event dataset, evaluated by its few-shot learning performance.

	9-way	3v3	11v11	6v6(Time warp)
TOQ-Net	**88.4**	**77.4**	**77.1**	**86.9**
STGCN-2D	84.5	17.5	16.6	39.7

Table A.2: Ablation study of the STGCN-2D model on the soccer event dataset, evaluated by its generalization to different number of players and time-warped trajectories.

of each node with this. This baseline replicates a very similar propagation rule as our relational reasoning layer. However, it still uses temporal convolutions to model temporal structures.

A.1.4 STGCN-2D

The original STGCN does not support input features of higher dimensions, so we extend STGCN to STGCN-2D to add binary inputs such as the distance. This extension slightly improves the model performance on the standard 9-way classification test. It also helps in the few-shot setting. Specifically, on the few-shot learning setting:

Meanwhile, adding extra binary inputs does not improve the generalization of the network to games with a different number of agents or to time-warped sequences. This following table extends the Table 2 in the main text. Specifically, we train STGCN-2D on 6v6 soccer games and test it on scenarios with a different number of agents (3v3 and 11v11) and also temporally warped trajectories. Our model shows a significant advantage.

A.1.5 STGCN-LSTM

The STGCN-LSTM model first encodes the input trajectory with an STGCN module. The output of the STGCN module is a tensor of shape $T \times N \times 256$, where T is the length of the input trajectory, and N is the number of entities. We perform an average pooling over the entity dimension and get a tensor of shape $T \times 256$. Next, we encode this feature with a 2-layer bidirectional LSTM module. The hidden dimension for the LSTM module is 256. We use a single softmax layer on top of the LSTM module to

Model	Reg.	1-Shot	Full
STGCN	$99.89_{\pm 0.05}$	$94.92_{\pm 1.03}$	$98.79_{\pm 0.23}$
STGCN-LSTM (kernel size 3)	$99.89_{\pm 0.05}$	$96.16_{\pm 1.39}$	$99.04_{\pm 0.31}$
STGCN-LSTM (kernel size 1)	$99.92_{\pm 0.03}$	$95.48_{\pm 1.67}$	$98.86_{\pm 0.43}$
TOQ-Net	$\mathbf{99.96}_{\pm 0.02}$	$\mathbf{98.04}_{\pm 0.97}$	$\mathbf{99.48}_{\pm 0.24}$

Table A.3: Ablation study of different kernel sizes for the STGCN-LSTM model on the RLBench dataset, measured by per-action (macro) accuracy and averaged of four 1-shot splits and four random seeds per split. The ± values indicate standard errors.

classify the action.

STGCN-LSTM uses the same architecture as STGCN for both tasks, except for the kernel size and the stride, because our goal is to evaluate the performance of recurrent neural networks (the LSTM model) on modeling temporal structures. In the soccer event detection task, we use kernel size 3 and stride 1 so that the STGCN module will have the representational power to encode local physical properties. In the manipulation concept learning task, we use a temporal kernel size of 1 and stride 1. Meanwhile, Table A.3 shows the ablation study of different kernel sizes, evaluated by the few-shot learning performance on the RLBench dataset. Different kernel sizes (1 and 3) have a similar performance.

A.1.6 Space-Time Region Graph Networks

We use the same architecture for each Space-Time layer as Wang et al. proposed in Space-Time Graph [Wang and Gupta, 2018]. Table A.5 summarises the detailed parameters of all layers.

A.1.7 Non-Local Neural Networks

We use the same architecture for each Non-local layer as Wang et al. proposed in Non-Local networks [Wang *et al.*, 2018]. The original non-local network works on pixels. Here instead, we apply it to the space-time graph constructed in the same way as Wang and Gupta [2018]. The major difference between the Space-Time Graph and the Non-local Network is that Space-Time Graph uses graph convolutions while Non-local uses transfomer-like attentions over all nodes in the graph. Table A.6 summarises the detailed parameters of all layers.

Soccer event detection. We use the identical training strategy for our model and both baselines. Each training batch contains 128 examples, which are sampled as following: we first uniformly sample an action category, and then uniformly sample a trajectory labeled as this action category.

In all the soccer event tasks, we optimized our model using Adam [Kingma and Ba, 2015], with the learning rate initialized to $\eta = 0.002$. The learning rate is fixed for the first 50 epochs. After that, we decrease the learning rate η by a factor of 0.9 if the best validation loss has not been updated during the last 6 epochs.

All models are trained for 200 epochs. In the task of generalization to different input scales, we first train different models on the original dataset for 200 epochs ($\sim$40,000 iterations) and finetune them on the new dataset with few samples for 10,000 iterations.

Manipulation concept learning. We optimized different models using Adam with the initial learning rate $\eta = 0.003$. We applied the same learning rate schedule as in the soccer event detection task. The batch size for the manipulation concept learning task is 32 for models with STGCN backbone and is 4 for TOQ-Nets. All models are trained for 150 epochs.

Model	Input Dim.	Output Dim.	Kernel Size	Stride	Dropout	Residual
STGCN	-	64	7	1	0	False
	64	64	7	1	0.5	True
	64	64	7	1	0.5	True
	64	64	7	1	0.5	True
	64	128	7	2	0.5	True
	128	128	7	1	0.5	True
	128	128	7	1	0.5	True
	128	256	7	2	0.5	True
	256	256	7	1	0.5	True
	256	256	7	1	0	True
STGCN_S	-	16	7	1	0	False
	16	16	7	1	0.5	True
	16	32	7	2	0.5	True
	32	32	7	1	0.5	True
	32	64	7	2	0.5	True
	64	128	7	1	0	True
STGCN_T	-	8	7	1	0	False
	8	12	7	2	0.5	True
	12	64	7	1	0	True

Table A.4: The STGCN architecture used in the paper.

Model	Input Dim.	Output Dim.	Stride	Residual
Space-Time	-	64	1	False
	64	64	1	True
	64	64	1	True
	64	64	1	True
	64	128	2	True
	128	128	1	True
	128	128	1	True
	128	256	2	True
	256	256	1	True
	256	256	1	True
Space-Time$_S$	-	16	1	False
	16	16	1	True
	16	32	2	True
	32	64	2	True
	64	128	1	True
Space-Time$_T$	-	32	1	False
	32	64	2	True
	64	64	2	True

Table A.5: The Space-Time Region Graph architecture used in the paper.

Model	Input Dim.	Output Dim.	Stride	Residual
Non-Local	-	64	1	False
	64	64	1	True
	64	64	1	True
	64	64	1	True
	64	128	2	True
	128	128	1	True
	128	128	1	True
	128	256	2	True
	256	256	1	True
	256	256	1	True
Non-Local$_S$	-	32	1	False
	32	32	2	True
	32	64	2	True
	64	128	1	True
Non-Local$_T$	-	32	1	False
	32	32	2	True
	32	64	2	True

Table A.6: The Non-local Neural Networks architecture used in the paper.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Details and Results for RatSkills

This appendix chapter provides details for RatSkills introduced in Chapter 4. In Appendix B.1, we present the experimental setups, in particular the dataset splits and features. In Appendix B.2 we discuss the implementation details of our model RatSkills and other baselines.

B.1 Dataset

B.1.1 Crafting World

Our Crafting World environment is based on the Crafting environment introduced by Chen *et al.* [2021]. The environment has a crafting agent that can move in a grid world, collect resources, and craft items. In the original environment, every crafting rule is associated with a unique crafting station (e.g., paper must be crafted on the paper station). We modified the rules such that some crafting rules can share a common crafting station (e.g., both arrows and swords can be crafted on a weapon station). We add additional tiles: doors and rivers into the environment. Toggling a specific switch will open all doors. Otherwise, the agent can move across doors when they are holding a key. Meanwhile, the agent can move across rivers when they have a boat in their inventory.

We have used 47 object types in Crafting World including obstacles (e.g., river tiles, doors), items (e.g., axe), resources (e.g., trees), and crafting stations. We use 27 rules for mining resources and crafting items. When the agent is at the same tile as another object, the *toggle* action will trigger the object-specific interaction. For item, the *toggle* action will pick up the object. For resource, the *toggle* action will mine the resource if the agent has space in their inventory and has the required tool for mining this type of resource (e.g., pickaxe is needed for mining iron ore).

State representation. The state representation of Crafting World consists of three parts.

1. The *global feature* contains the size of grid world, the location of the agent, and the inventory size of the agent.
2. The *inventory* feature contains an unordered list of objects in the agent's inventory. Each of them is represented as a one-hot vector indicating its object type.
3. The *map* feature contains all objects on the map, including obstacles, items, resources, and crafting stations. Each of them is represented by a one-hot type encoding, the location (as integer values), and state (e.g., *open* or *closed* for doors).

Action. In Crafting World, there are 5 primitive level actions: *up*, *down*, *left*, *right*, and *toggle*. The first four actions will move the agent in the environment, while the *toggle* action will try to interact with the object in the same cell as the agent.

State feature extractor. Since our state representation contain a varying number of objects, we extract a vector representation of the environment with a relational neural network: Neural Logic Machines [NLM; Dong *et al.*, 2019].

Concretely, we extract the inventory feature and the map feature separately. For each item in the inventory, we concatenate its input representation (i.e., the object type) with the global input feature. We process each item with the same fully-connected layer with ReLU activation. Following NLM [Dong *et al.*, 2019], we use a max pooling operation to aggregate the feature for all inventory objects, resulting in a 128-dim vector. We use a similar architecture (but different neural network weights) to process all objects on the map. Finally, we concatenate the extracted inventory feature (128-dim), the map feature (128-dim), and the global feature (4-dim) as the holistic state representation. Thus, the output feature dimension for each state is 260.

Task definitions. We list the task descriptions in the *primitive*, the *compositional*, and the *novel* splits Table B.1.

Primitive	
grab pickaxe	grab axe
grab key	toggle switch
craft wood plank	craft stick
craft shears	craft bed
craft boat	craft sword
craft arrow	craft cooked potato
craft iron ingot	craft gold ingot
craft bowl	craft beetroot soup
craft paper	mine gold ore
mine iron ore	mine sugar cane
mine coal	mine wood
mine feather	mine wool
mine potato	mine beetroot

Compositional
grab pickaxe
grab axe
grab key
toggle switch
mine wood **then** craft wood plank
craft wood plank **then** craft stick
craft iron ingot **or** craft gold ingot **then** craft shears
mine wool **and** craft wood plank **then** craft bed
craft wood plank **then** craft boat
craft iron ingot **and** craft stick **then** craft sword
mine feather **and** craft stick **then** craft arrow
mine potato **and** mine coal **then** craft cooked potato
mine iron ore **and** mine coal **then** craft iron ingot
mine gold ore **and** mine coal **then** craft gold ingot
craft wood plank **or** craft iron ingot **then** craft bowl
craft bowl **and** mine beetroot **then** craft beetroot soup
mine sugar cane **then** craft paper
grab pickaxe **then** mine gold ore
grab pickaxe **then** mine iron ore
grab pickaxe **or** grab axe **then** mine sugar cane
grab pickaxe **then** mine coal
grab axe **then** mine wood
craft sword **then** mine feather
craft shears **or** craft sword **then** mine wool
grab axe **or** mine coal **then** mine potato
grab axe **or** grab pickaxe **then** mine beetroot

Novel
1. mine sugar cane **then** craft paper
2. mine potato **and** (gran pickaxe then mine coal) **and** craft cooked potato
3. mine beetroot **and** (grab axe then mine wood then craft wood plank then craft bowl) **then** craft beetroot soup
4. grab axe **then** mine wood **then** craft wood plank **then** grab pickaxe **then** mine iron ore **and** mine coal **then** craft iron ingot **then** craft shears **then** mine wool **then** craft bed
5. grab axe **then** mine wood **then** craft wood plank **then** craft stick **then** grab pickaxe **then** mine iron ore **and** mine coal **then** craft iron ingot **then** craft sword **then** mine feather **then** mine wood **then** craft wood plank **then** craft stick **then** craft arrow
6. grab key **then** grab axe
7. toggle switch **then** mine beetroot
8. grab axe **then** mine wood **then** craft wood plank **then** craft boat **then** mine sugar cane
9. grab axe **then** mine wood **then** craft wood plank **then** craft boat **then** grab pickaxe
10. grab key **then** grab axe **then** mine wood **then** craft wood plank **then** craft boat **then** mine potato
11. grab key **or** (grab axe **then** mine wood **then** craft wood plank **then** craft boat) **then** grab pickaxe **then** mine gold ore
12. grab axe **then** mine wood **then** craft wood plank **then** craft boat **then** grab key **or** toggle switch **then** grab pickaxe **then** mine iron ore **and** mine coal **then** craft iron ingot

Table B.1: Task descriptions in the *primitive*, *compositional* and *novel* sets for the Crafting World.

B.1.2 Playroom

We build our Playroom environment following Konidaris *et al.* [2018]. Specifically, we have added obstacles into the environment. The environment contains an agent, 6 effectors (a ball, a bell, a light switch, a button to turn on the music, a button to turn off the music and a monkey), and a fix number of obstacles. The agent and the effectors have fixed shapes. Thus, their geometry can be fully specified by their location and orientation. For simplicity, we have also fixed the shape and the location of the obstacles.

State representation. We represent the pose of the agent by a 3D vector including the x, y coordinates (real-valued) and its rotation (real-valued, in $[-\pi, \pi)$. The state

representation consist of the pose of the agent (as a 3-dimensional vector) and the locations of six effectors (as 6 2-dimensional vectors). Note that the state representation does not contain the shapes nor the locations of obstacles as they remain unchanged throughout the experiment. We directly concatenate these 7 vectors as the state representation.

Action. The agent has a 3-dimensional action space: $[-1, 1]^3$. That is, for example, at each time step, the agent can at most move 1 meter along the x axis. We perform collision checking when the agent is trying to make a movement. If an action will result in a collision with objects or obstacles in the environment, the action will be treated as invalid and the state of the agent will not change.

Task definitions. We list the task descriptions in each of the *primitive*, *compositional* and *novel* set of the Playroom in Table B.2

Primitive		
play the ball	ring the bell	turn on the light
touch the mounkey	turn off the music	turn on the music
Compositional (designed meaningful tasks)		
play the ball		
turn on the light **then** ring the bell		
turn on the music **and** play the ball **then** touch the monkey		
play the ball **then** turn on the light		
turn on the music **and** play the ball **then** turn off the music		
turn on the music **or** play the ball		
turn off the music **then** play the ball **then** turn on the music		
turn on the music **and** play the ball **and** turn on the light **then** ring the bell		
Novel (randomly sampled)		
play the ball **then** turn on the light **or** ring the bell		
turn on the music **then** turn on the light		
turn on the music **then** turn on the light		
play the ball **then** touch the monkey		
turn on the music **then** turn off the music		
turn on the music **and** ring the bell **then** touch the monkey		
ring the bell **then** touch the monkey **then** turn on the light		
turn on the light **and** (ring the bell **or** turn on the music) **then** play the ball		

Table B.2: Task descriptions in the *primitive*, *compositional* and *novel* sets for the Playroom.

B.2 Implementation Details

In this section, we present the implementation details of RatSkills and other baselines. Without further notes, through out this section, we will be using the same LSTM encoder for task descriptions in $\mathcal{TL}$, and the same LSTM encoder for state sequences. The architecture of both encoders will be presented in Appendix B.2.1.

B.2.1 LSTM

Task description encoder. We use a bi-directional LSTM [Hochreiter and Schmidhuber, 1997] with a hidden dimension of 128 to encode the task description. The vocabulary contains all primitive skills, parentheses, and three connectives (*and*, *or*, and *then*). We perform an average pooling on the encoded feature for both directions, and concatenate them as the encoding for the task description. Thus, the output dimension is 256.

State sequence encoder. For a given state sequence $\bar{s} = \{s_i\}$, we first use a fully-connected layer to map each state s_i into a 128-dimensional vector. Next, we feed the sequence into a bi-directional LSTM module. The hidden dimension of the LSTM is 128. We perform an average pooling on the encoded feature for both directions, and concatenate them as the encoding for the state sequence.

Training. In our LSTM baseline for inverse planning, we concatenate the state sequence feature and the task description feature, and use a 2-layer multi-layer perceptron (MLP) to compute the score of the input tuple: (trajectory, task description). The LSTM model is trained for 100 epochs on both environments. Each epoch contains 30 training batches that are randomly sampled from training data. The batch size is 32. We use the RMSProp optimizer with a learning rate decay from 10^{-3} to 10^{-5}.

B.2.2 Inverse Reinforcement Learning (IRL)

The IRL baseline uses an LSTM model to encode task descriptions. We use different parameterizations for the reward function and the Q function in two datasets.

Crafting World Since the task description may have complex temporal structures, the reward value does not only condition on the current state and but all historical states. Therefore, instead of $Q(s, a|t)$ and $R(s, a, s'|t)$, we use $Q(\bar{s}, a|t)$ and $R(\bar{s}, a, s'|t)$ to parameterize the Q function and reward function, where s is the current state, a

the action, t the task description, s' the next state, and $\bar{s}$ the historical state sequence from the initial state to the current state.

We use neural networks to approximate the Q function and reward function. For both of them, $\bar{s}$ is first encoded by an LSTM model into a fixed-length vector embedding. We simply concatenate the historical state encoding and the task description encoding, and then use a fully-connected layer to map the feature into a 5-dimensional vector. Each entry corresponds to the Q value or the reward value for a primitive action.

Playroom The Q function and reward function in Playroom also condition on all historical states. In Playroom, we parameterize the value of each state: $V(\bar{s})$, instead of $Q(\bar{s}, a)$. We parameterize $R(\bar{s}, a, s')$ as $R(\bar{s}, s')$.

The input to our reward function network is composed of three parts: the vector encoding of the historical state sequence, the vector encoding for the next state s', and the task description encoding. We concatenate all three vectors and run a fully-connected layer with a logSigmoid activation function.*

In Playroom, since we do not directly parameterize the Q value for all actions in the continuous action space, in order to sample the best action at each state s for plan execution, we first randomly sample 20 valid actions from the action space (i.e., actions that do not lead to collision), and choose the action that maximizes the Q function: $Q(\bar{s} \cup s', a)$, where $\bar{s}$ is the historical state seuqnce and s' is the next state after taking a.

Value iteration. Both environments have a very large (Crafting World) or even infinite (Playroom) state space. Thus it is impossible to run value iteration on the entire state space. Thus, at each iteration, for a given demonstration trajectory $(\bar{s}_e, \bar{a}_e)$, we construct a self exploration trajectory $(\bar{s}_p, \bar{a}_p)$ that share the same start state as $\bar{s}_e$ †. We run value iteration on $\{\bar{s}_e\} \cup \{\bar{s}_p\}$. For states not in this set, we use the Q function network to approximate their values.

Training. For both Crafting World and Playroom, we train the IRL model for 60 epochs. We set the batch size to be 32 and each epoch has 30 training batches. We use a replay buffer that can store 100,000 trajectories. For both environments, we use the Adam optimizer with a learning rate decay from 10^{-3} to 10^{-5}. We have found the IRL method unstable to train in the Playroom environment. Thus, in Playroom, we

*We have experimented with no activation function, Sigmoid, and logSigmoid activations, and found that the logSigmoid activation works the best.

†Since running self-exploration in Playroom is too slow, in practice, we only generate self-exploration trajectories for 4 trajectories in the input batch.

use a warm-up training procedure. In the first 18(30%) epochs, we set $\gamma = 0$ for a "warm start", and for rest of the epochs we use $\gamma = 0.5$, where γ is the discount factor in the Q function.

B.2.3 Behavior Cloning (BC)

BC learns a policy $\pi(\bar{s}, a|t)$ from data, where t is the task description, a a primitive action, and $\bar{s}$ the historical state sequence. The state sequence $\bar{s}$ is first encoded by an LSTM model into a fixed-length vector embedding.

In Crafting World, we use a fully-connected layer with softmax activation to parameterize $\pi(a|\bar{s}, t)$. Specifically, the input to the fully-connected layer is the concatenation of the vector encoding of $\bar{s}$ and the vector encoding of the task description t.

In Playroom, we use two fully-connected (FC) layers to parameterize $\pi(a|\bar{s}, t)$. Specifically, we parameterize $\pi(a|\bar{s}, t)$ as a Gaussian distribution. The first FC layer has a Tanh activation and parameterizes the mean μ of the Gaussian. The second FC layer has a Sigmoid activation and paramerizes the standard variance σ^2 of the Gaussian.

To make this model more consistent with our BC-FSM model, in both environments, we also train a module to compute the termination condition of the trajectory. That is, a neural network that maps $\bar{s}$ to a real value in $[0, 1]$, indicating the probability of terminating the execution. Denote the output of this network as $\mathit{stop}(\bar{s})$. At each time step, the agent will terminate its policy with probability $\mathit{stop}(\bar{s})$. We modulate the probability for other actions a as $\pi(a|\bar{s}, t) \cdot (1 - \mathit{stop}(\bar{s}))$

For planning in Crafting World, at each step, we choose the action with the maximum probability (including the option to "terminate" the execution). In Playroom, we always take the "mean" action parameterized by $\pi(a|\bar{s}, t)$ until we reach the maximum allowed steps.

We then define the score of a task given a trajectory, $\mathit{score}(\bar{s}, \bar{a}, t)$, as the sum of log-probabilities of the actions taken at each step. We train this model with the same loss and training procedure as RatSkills. We train the model for 100 epochs using the Adam optimizer with a learning rate decay from 10^{-3} to 10^{-5}.

B.2.4 Behavior Cloning with fsm (BC-fsm)

BC-FSM represents task description as an FSM, in the same way as our model RatSkills. It represents each skill o as a tuple: $\langle \pi_o(sa), \mathit{stop}_o(s) \rangle$, corresponding to the skill-conditioned policy and the termination condition.

Inverse planning. The inverse planning procedure for BC-FSM jointly segments the trajectory and computes the consistency score between the task description and the input trajectory. In particular, our algorithm will assign an FSM state v_i to each state s_i, and insert several action. We use a dynamic programming algorithm (similar to the one used by our algorithm for RatSkills) to find the assignment that maximize the overall score:

$$score(\bar{s}, \bar{v}, \bar{a}) := \prod_i p(a_i|s_i, v_i, t)$$

$$p(a_i|s_i, v_i, t) = \begin{cases} \pi(a_i|s_i, v_i) \cdot (1 - stop(s_i, v_i)) & \text{if } a \in \mathcal{A} \text{ is a primitive action} \\ stop(s_i, v_i) & \text{if } a \in E_t \text{ is an FSM transition} \end{cases}$$

Planning. We use the same strategy as the basic Behavior Cloning model to choose actions at each step, conditioned on the current FSM state. BC-FSM handles branches in the FSM in the same way as our algorithm for RatSkills.

B.2.5 FSM-AStar

We have implemented a extended version of the A^* algorithm to handle FSM states in Crafting World.

***A^** at each fsm node.** We start with the A^* search process happening at each FSM node. For a given FSM state, the A^* search extends the tree search in two stages. The first stage lasts for $b = 3$ layers during training and $b = 4$ layers during testing. In the first b layers of the search tree, no pruning is applied to nodes. Thus, a single root node gets expanded up to 5^b nodes during the first stage. The second stage lasts for $c = 15$ layers during training and 25 layers in testing. In layer $d \in [b + 1, b + c]$, we perform pruning based on the heuristic for each node. For each FSM node v and each layer d, we only keep the top $k = 10$. Finally, we run the value iteration on the entire search tree.

To accelerate this search process, for all tasks t in the training set, we have initialized a dedicated value approximator $V_t(\bar{s})$, conditioned on the historical state sequence. During training, we use the value iteration result on the generated search tree to supervise the learning of this approximator V_t. Meanwhile, we use the value prediction of V_t as the heuristic function for node pruning. During test, since we may encounter unseen tasks, the A^*-FSM search uses a uniform heuristic function $h \equiv 0$

Search on an fsm. For a given initial state s_0 and task description t, we first build FSM_t and add the search tree node (s_0, v_0) to the search tree, where v_0 is the initial

node of the FSM. Then we expand the search tree nodes (s, v) by a topological order of v. It has two stages. First, for each FSM node v, we run up to 5000 steps of A^* search. Next, for all search tree nodes (s, v) at FSM node v, we try to make a transition from (s, v) to (s, v') where (v, v') is a valid edge in FSM_t. Finally, we output a trajectory ending at the FSM node v_T with minimum cost.

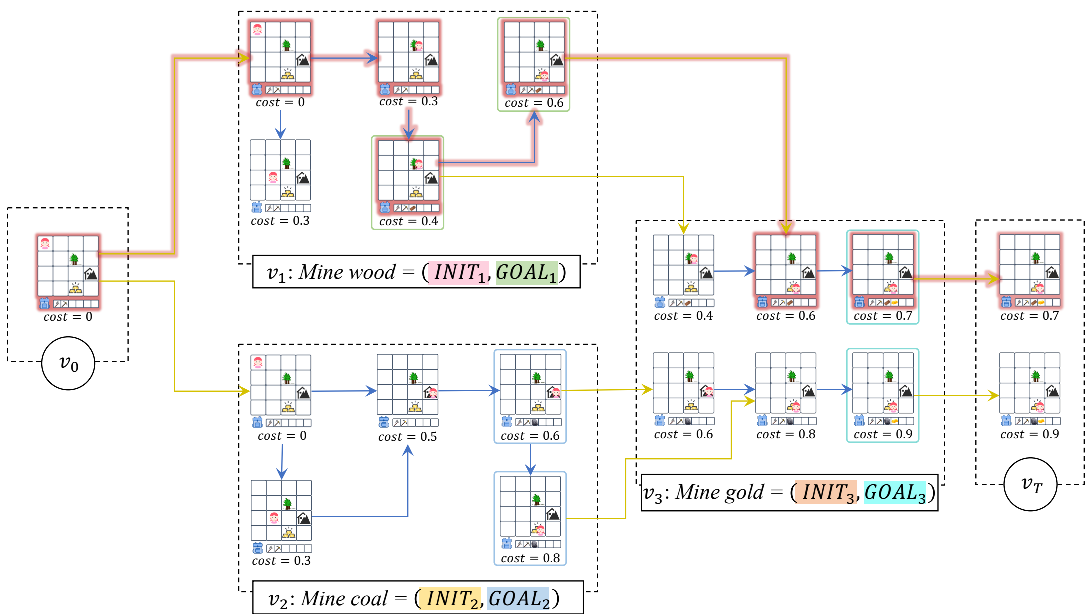


Figure B-1: A running example of the FSM-A^* algorithm for the task "*(mine wood* ***or*** *mine coal)* ***then*** *mine gold.*" For simplicity, we only show a subset of states visited on each FSM node. The blue arrows indicate transitions by primitive actions (in this example, each primitive action takes a cost of 0.1). The yellow arrows are transitions on the FSM, which can only be performed when $G_v(\cdot)$ and $I_{v'}(\cdot)$ evaluates to True (in practice, the reward is computed as $-(\log \Pr(G_v(\cdot) + \log \Pr(I_{v'}(\cdot))))$. At the super-terminal node v_T, the state with minimum cost will be selected and we will back-trace the entire state-action sequence.

Example. Fig. B-1 shows a running example of our FSM-A^* planning given the task "*mine wood* ***or*** *mine coal* ***then*** *mine gold*" from the state s_0 (shown as the left-most state in the figure).

1. At the beginning, (s_0, v_0) is expanded to the node v_1:*mine wood* and v_2:*mine coal* with FSM transition actions at no cost.

2. We expand the search tree node on v_1 and v_2 and compute the cost for reaching each states on v_1 and v_2.

3. For states that satisfy the goal conditions for v_1 and v_2 (i.e., G_1 and G_2, respectively, and circled by green and blue boxes) and the initial condition for v_3 (i.e., I_3), we make a transition to v_3 at no cost (the states that do not satisfy the conditions can also be expanded to v_3 but with a large cost.

4. Then search can be done in a similar way at v_3 and the states at v_3 that satisfy G_3 can reach v_T.

5. For all states at v_T, we back-trace the state sequence with the minimum cost.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

Jacob Andreas and Dan Klein. Alignment-Based Compositional Semantics for Instruction Following. In *EMNLP*, 2015.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning With Policy Sketches. In *ICML*, 2017.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A Survey of Robot Learning From Demonstration. *Rob. Auton. Syst.*, 57(5):469–483, 2009.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*, 2015.

Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action Understanding as Inverse Planning. *Cognition*, 113(3):329–349, 2009.

Chris L Baker, Julian Jara-Ettinger, Rebecca Saxe, and Joshua B Tenenbaum. Rational Quantitative Attribution of Beliefs, Desires and Percepts in Human Mentalizing. *Nature Human Behaviour*, 1(4):1–10, 2017.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational Inductive Biases, Deep Learning, and Graph Networks. *arXiv:1806.01261*, 2018.

Calin Belta. Formal Synthesis of Control Strategies for Dynamical Systems. In *CDC*, 2016.

Christopher Bradley, Adam Pacheck, Gregory J. Stein, Sebastian Castro, Hadas Kress-Gazit, and Nicholas Roy. Learning and Planning for Temporally Extended Tasks in Unknown Environments. In *ICRA*, 2021.

William Brendel, Alan Fern, and Sinisa Todorovic. Probabilistic Event Logic for Interval-Based Event Recognition. In *CVPR*, 2011.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*, 2014.

Alberto Camacho, Jorge Baier, Christian Muise, and Sheila McIlraith. Finite LTL Synthesis as Planning. In *ICASP*, 2018.

Valerie Chen, Abhinav Gupta, and Kenneth Marino. Ask Your Humans: Using Human Instructions to Improve Generalization in Reinforcement Learning. In *ICLR*, 2021.

Sonia Chernova and Manuela Veloso. Confidence-Based Policy Learning From Demonstration Using Gaussian Mixture Models. In *AAMAS*, 2007.

Glen Chou, Necmiye Ozay, and Dmitry Berenson. Explaining Multi-Stage Tasks by Learning Temporal Logic Formulas From Suboptimal Demonstrations. In *RSS*, 2020.

Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. Modular Networks for Compositional Instruction Following. In *NAACL*, 2021.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM CSUR*, 33:374–425, 2001.

Srijan Das, Rui Dai, Michal Koperski, Luca Minciullo, Lorenzo Garattoni, Francois Bremond, and Gianpiero Francesca. Toyota Smarthome: Real-World Activities of Daily Living. In *ICCV*, 2019.

Zhiwei Deng, Arash Vahdat, Hexiang Hu, and Greg Mori. Structure Inference Machines: Recurrent Neural Networks for Analyzing Relations in Group Activity Recognition. In *CVPR*, 2016.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural Logic Machines. In *ICLR*, 2019.

Staffan Ekvall and Danica Kragic. Robot Learning From Demonstration: A Task-Level Planning Approach. *IJARS*, 5(3):33, 2008.

Richard Evans and Edward Grefenstette. Learning Explanatory Rules From Noisy Data. *JAIR*, 61:1–64, 2018.

Jerry Fodor and Brian P McLaughlin. Connectionism and the Problem of Systematicity: Why Smolensky's Solution Doesn't Work. *Cognition*, 35(2):183–204, 1990.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition*, 28(1-2):3–71, 1988.

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning Probabilistic Relational Models. In *IJCAI*, 1999.

Naresh Gupta and Dana S Nau. On the Complexity of Blocks-World Planning. *Artif. Intell.*, 56(2-3):223–254, 1992.

Robert F Hadley. Systematicity in Connectionist Language Learning. *Mind Lang.*, 9(3):273–287, 1994.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

Malte Helmert. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26:191–246, 2006.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *Signal Process.*, 29(6):82–97, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

Mostafa S. Ibrahim, Srikanth Muralidharan, Zhiwei Deng, Arash Vahdat, and Greg Mori. A Hierarchical Deep Temporal Model for Group Activity Recognition. In *CVPR*, 2016.

Stephen S Intille and Aaron F Bobick. A Framework for Recognizing Multi-Agent Action From Visual Evidence. In *AAAI*, 1999.

Stephen S Intille and Aaron F Bobick. Recognizing Planned, Multiperson Action. *CVIU*, 81(3):414–445, 2001.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The Robot Learning Benchmark & Learning Environment. *RA-L*, 2020.

Peter A Jansen and Scott Watter. Strong Systematicity Through Sensorimotor Conceptual Grounding: An Unsupervised, Developmental Approach to Connectionist Sentence Processing. *Connect. Sci.*, 24(1):25–55, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

Thomas N Kipf and Max Welling. Semi-Supervised Classification With Graph Convolutional Networks. In *ICLR*, 2017.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot Learning From Demonstration by Constructing Skill Trees. *IJRR*, 31(3):360–375, 2012.

George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *JAIR*, 61:215–289, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification With Deep Convolutional Neural Networks. In *NIPS*, 2012.

Luis C. Lamb, Rafael V. Borges, and A. d'Avila Garcez. A Connectionist Cognitive Model for Temporal Synchronisation and Learning. In *AAAI*, 2007.

Steven M LaValle et al. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, Computer Science Department, Iowa State University, 1998.

Dianhuan Lin, Eyal Dechter, Kevin Ellis, Joshua B Tenenbaum, and Stephen H Muggleton. Bias Reformulation for One-Shot Function Induction. In *ECAI*, 2014.

Zhezheng Luo, Jiayuan Mao, Jiajun Wu, Tomás Lozano-Pérez, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. Planning With Skills From Rational Demonstrations. In *IJCAI*, 2021.

David Maier and David S. Warren. *Computing With Logic: Logic Programming With Prolog.* Benjamin-Cummings Publishing Co., Inc., 1988.

Jiayuan Mao, Zhezheng Luo, Chuang Gan, Joshua B. Tenenbaum, Jiajun Wu, Leslie Pack Kaelbling, and Tomer D. Ullman. Temporal and Object Quantification Networks. In *IJCAI*, 2021.

Peter Ondruska Markus Wulfmeier and Ingmar Posner. Maximum Entropy Deep Inverse Reinforcement Learning. In *NeurIPS Workshop*, 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(7540):529–533, 02 2015.

Stephen Muggleton. Inductive Logic Programming. *New Gener. Comput.*, 8(4):295–318, 1991.

Stephen Muggleton. Stochastic Logic Programs. *Advances in Inductive Logic Programming*, 32:254–264, 1996.

Daniel Neider and Ivan Gavran. Learning Linear Temporal Properties. In *FMCAD*, 2018.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for Inverse Reinforcement Learning. In *ICML*, volume 1, page 2, 2000.

Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and Generalization of Complex Tasks From Unstructured Demonstrations. In *IROS*, 2012.

Nils J Nilsson. *Principles of Artificial Intelligence.* Springer Science & Business Media, 1982.

Ronald Parr and Stuart Russell. Reinforcement Learning With Hierarchies of Machines. In *NeurIPS*, 1998.

Leo de Penning, A. d'Avila Garcez, Luís C. Lamb, and John-Jules C. Meyer. A Neural-Symbolic Cognitive Agent for Online Learning and Reasoning. In *IJCAI*, 2011.

Amir Pnueli. The Temporal Logic of Programs. In *FOCS*, 1977.

Mengshi Qi, Jie Qin, Annan Li, Yunhong Wang, Jiebo Luo, and Luc Van Gool. stagNet: An Attentive Semantic RNN for Group Activity Recognition. In *ECCV*, 2018.

Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent Advances in Robot Learning From Demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330, 2020.

Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, and Santosh Srivastava. Logical Neural Networks. In *NeurIPS*, 2020.

Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A Learning Based Approach to Control Synthesis of Markov Decision Processes for Linear Temporal Logic Specifications. In *CDC*, pages 1091–1096, 2014.

Ankit Shah, Pritish Kamath, Julie A Shah, and Shen Li. Bayesian Inference of Temporal Task Specifications From Demonstrations. In *NeurIPS*, 2018.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the Game of Go Without Human Knowledge. *Nature*, 550(7676):354, 2017.

Jürgen Stränger and Bernhard Hommel. The Perception of Action and Movement. In *Handbook of Perception and Action*, volume 1, pages 397–451. Elsevier, 1996.

Shao-Hua Sun, Te-Lin Wu, and Joseph J Lim. Program Guided Agent. In *ICLR*, 2020.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning With Neural Networks. In *NIPS*, 2014.

Kevin Tang, Li Fei-Fei, and Daphne Koller. Learning Latent Temporal Structure for Complex Event Detection. In *CVPR*, 2012.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. In *AAAI*, 2011.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral Cloning From Observation. In *IJCAI*, 2018.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Teaching Multiple Tasks to an RL Agent Using LTL. In *AAMAS*, 2018.

Son D Tran and Larry S Davis. Event Modeling and Recognition Using Markov Logic Networks. In *ECCV*, 2008.

Robin R Vallacher and Daniel M Wegner. What Do People Think They're Doing? Action Identification and Human Behavior. *Psychol. Rev.*, 94(1):3, 1987.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NIPS*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In *NIPS*, 2015.

Xiaolong Wang and Abhinav Gupta. Videos as Space-Time Region Graphs. In *ECCV*, 2018.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-Local Neural Networks. In *CVPR*, 2018.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation. *arXiv:1609.08144*, 2016.

Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition. In *AAAI*, 2018.

Jeffrey M Zacks, Barbara Tversky, and Gowri Iyer. Perceiving, Remembering, and Communicating Structure in Events. *J. Exp. Psychol. Gen.*, 130(1):29, 2001.

Jeffrey M Zacks, Nicole K Speer, Khena M Swallow, Todd S Braver, and Jeremy R Reynolds. Event Perception: A Mind-Brain Perspective. *Psychol. Bull.*, 133(2):273, 2007.

Tan Zhi-Xuan, Jordyn L Mann, Tom Silver, Joshua B Tenenbaum, and Vikash K Mansinghka. Online Bayesian Goal Inference for Boundedly-Rational Planning Agents. In *NeurIPS*, 2020.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, 2008.