

# *Abstract*

## **Interactive Reinforcement Learning from Human Language and Evaluative Feedback Through Task Decomposition**

by Guan ‘Royal’ WANG, Ph.D., Brown University, August 2020

In this work, I present an implemented model that can learn interactively from natural language, enabling non-expert human trainers to convey complex tasks to machines using task decomposition and a combination of evaluative feedback and natural language.

Over a series of experiments and associated algorithms, I show:

1. Contrary to the assumptions of methods that learn behavior interactively, feedback from people is policy dependent.
2. The method of COnvergent Actor-Critic by Humans (COACH) can interpret and learn from this kind of feedback.
3. Complex tasks can be hard to learn directly from feedback, but non-expert human trainers can decompose complex tasks into simpler units.
4. Geometric Linear Temporal Logic (GLTL) can be used as a logical form that can capture decomposed task descriptions and serve as the basis of an effective learning algorithm for building up complex tasks.
5. People can use natural language feedback to convey evaluative feedback effectively.
6. A deep sequence-to-sequence (Seq2Seq) approach can be used to interpret natural language feedback while discovering how to convert spontaneous natural language input to GLTL for machine execution.

These elements are demonstrated in an implemented system that learns online from interaction with an end user to interpret and execute the user’s tasks.



# **Interactive Reinforcement Learning from Human Language and Evaluative Feedback Through Task Decomposition**

Guan "Royal" Wang  
B. E. , Shanghai Jiao Tong University, 2011  
Sc. M., Brown University, 2015

A dissertation submitted in partial fulfillment of the  
requirements for the Degree of Doctor of Philosophy  
in the Department of Computer Science at Brown University

Providence, Rhode Island

August 2020



This dissertation by Guan Wang is accepted in its present form by  
the Department of Computer Science as satisfying the dissertation requirement  
for the degree of Doctor of Philosophy.

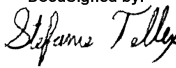
3/31/2021  
Date \_\_\_\_\_



Michael L. Littman, Advisor

Recommended to the Graduate Council

3/26/2021  
Date \_\_\_\_\_

DocuSigned by:  
  
D31243E8557A447...


Stefanie A Tellex, Reader

3/26/21  
Date \_\_\_\_\_



Daniel C Ritchie, Reader

3/26/2021  
Date \_\_\_\_\_

DocuSigned by:  
  
5BAB23154DF345D...

Ellie Pavlick, Reader

Approved by the Graduate Council

Date \_\_\_\_\_

Andrew G. Campbell, Dean of the Graduate School

## VITA

Guan "Royal" Wang was born in Liaoning, China. He began his story working with computers at the age around ten when he received his first Windows-98 desktop and has not stopped since. After graduating from Dalian No. 24 High School, Royal went to Shanghai Jiao Tong University in 2007. He received his B.E. from the Department of Automation, the School of Electronic Information and Electrical Engineering in 2011. During the undergraduate study, Royal spent time as a research intern at the machine-learning group at Microsoft Research, where he worked on web-user-behavior analytics and intent prediction. He then went to University of Washington in Seattle, WA as a visiting scientist to do research in human computer interaction. Royal started to work with Prof. Michael L. Littman as a Ph.D student at Brown University in Providence, RI in 2015 after receiving his Sc. M. degree of Computer Science from Brown. He focused on interactive reinforcement learning with humans in the loop. He co-founded Learnable.ai, an artificial intelligence startup company, to apply state-of-the-art and innovative machine-learning techniques to develop effective AI solutions that can help people live better lives.

*“To my family. To the people I love.”*





## *Acknowledgements*

My Ph.D. journey has been filled with challenges and surprise, a lot of which are beyond imagination. It may be possible for some other people to achieve the goals alone, but I'm not one of them. I would like to acknowledge people who have supported me, helped me, taught me, sponsored me, encouraged me, inspired me, pushed me, and worked with me together to make what I believe come true. I extend my deepest gratitude to:

- My family. My parents, Lijun Qi and Fuxin Wang, give me the freedom I need to spend as much time as possible to focus on the things that I love. They never stopped worrying about me for both my life and study, making me feel that learning from experienced human teachers is not only necessary but also useful. That feeling is very interesting, and may have had influence on some of the research ideas in this work. My parents provided me with an environment in which almost all the noise of life is filtered out, and I can do whatever I want as long as it's a good thing in a socially correct way. They never pushed me to work as a way to make money, the only thing they cared is whether what I do makes me a good person. As the only child of a family, I am so lucky to have my parents respect and believe in knowledge and truth, even though they could not understand my work.
- My advisor. Prof. Michael Lederman Littman, has given me both positive and negative feedback during the five years that we worked together. Some of his feedback were policy independent, for example, he taught me to always focus on the detailed experimental data and figures regardless of how complex the work is. Michael also insisted on understanding the entire pipeline instead of only looking at the results. In a word, "What", "How", and "Why" all matter. Some of his feedback was policy dependent, in that he encouraged me to try more of my own ideas over time, more exploration over exploitation as I dove deeper in the research area. I think I've got a lot of negative feedback as well, especially on how to organize a research team, how to write things out and submit in time. I was at a crossroad in my life when Michael met me, he made my days at Brown colourful again. He showed me how curiosity, passion, hard working, communication, and teamwork could make a person outstanding. I feel so happy to work with him and receive his advice, and, if I'm lucky, his influence will stay with me in my work for a very long time.
- My labmates during my time at Brown. I could always get help and support from: Mark Ho, Carl J Trimbach, Jun Ki Lee, David Abel, Sam Saarinen, Lucas Lehnert, Stephen Brawner, James MacGlashan, Evan Cater, Kavosh Asadi. People outside RLAB were also very helpful: Nakul Gopalan, Ifrah Idrees, Roma Patel.
- My teammates at Brown. I'm very proud and happy to have worked with a wonderful group of talented young researchers who joined my team and contributed to different parts of my research: Yiheng Xie, Andrew Yun Ho Kim, Troy Prebenda, Sahdiah Cox, Jordan Hartzell (interactive teaching robot), Jared Cohen, Jiyun Lee, Philip Strauss, Tushar Bhargava (internet of things), Nate Bowditch, Ben Most (interactive math tutor).
- My extended collection of research associates. Scientists that made direct contributions to my research: Robert Loftin, Bei Peng, David L. Roberts, Matthew E. Taylor (interactive reinforcement learning), Jeff Huang, Alexandra Papoutsaki, Hua Guo, Connor Gramazio, Jeff Rasley, Danae Metaxa-Kakavouli, Wenting Xie (human computer interaction), Blase Ur, Tim Nelson, Enrique A Areyan Viqueira (internet of things), Musik Kwon, Alex Hills, Robert C Zeleznik, Rosemary Simpson (interactive math

tutor), Stefanie A Tellex, Ellie Pavlick, Daniel C Ritchie, Leslie Kaelbling (interactive reinforcement learning with natural language). I also want to thank Andries van Dam, who was the reason I chose to come to Brown. Andy and I may have different opinions, but I always respect and appreciate his advice and help.

- The Brown staff. The professionals kept things organized so that my research could go well: Peter Haas made it possible for the experiments with the interactive teaching robot to happen by offering devices and managing spaces. Lori Agresti helped manage the research costs, Suzanne M Alden, Kathy Kirman Billings, Lisa J Manekofsky helped me with the other financial support. The technical staff (T-staff) John Bazik, Benjamin E Nacar, Phirum Peang maintained the cluster and my devices. Frank Pari helped me design and build my servers, saved my workstations several times, and offered a lot more personal help. Cynthia Ellis and Carol Cohen helped me mentally when I was down. Lauren K Clarke was the person that I would always talk to if there were still problems that could not be solved by the other people.
- My friends. I often turn to my friends for their knowledge, skills, and suggestions: Geng Ji, Yan Li, Li Sun, Jing Qian, Yin (Frank) Yu, Junyang Chen, Yuhang Song, Chris Tanner, Qi Xin, Cheng Xie, Zhile Ren, Da Yu, Zhiqiang Sui, Guangyao Zhou, Takehiro Oyakawa, Do Kook Choe (D. K.), Nedi Daskalova, Xinyi Zhang, Xuan Zhang, Dongkun Zhang, Xu Zhang, Ziyang Liu, Xiaotong Wang.
- My Learnable team. The people who make amazing things happen: Lan, Jack, Lionel, Meng, Haotian, Matt, Eugene, Steve, Aaron, John, Richard, Clara, Mr. Jin, Mr. Yi, Dr. Zhao, Xin, Xuesong, Evan, Fang, Hua, Prof. Yu, Prof. Yang, Prof. Chu, Tianqi, Elenor, Ran, Hongwu, Ziheng, Jesse, Claire, Chi, Victor, Ziyi, Tianyang, Rahul, Prem, Mason, Xiaoling, Qijia, Bao, and many more.
- My sponsors. My research has been supported by DARPA XAI, DARPA L2M, DARPA LwLL programs, Samsung, National Science Foundation, grants from the Department of Computer Science, grants from Brown University, prizes from Massachusetts Institute of Technology, awards from Harvard University, and a few more generous sponsors.
- My dog McQueen. We met each other at the beginning of both my Ph.D. journey and his life, and we've stayed together ever since. McQueen inspired me to think about how a dog learns and behaves, which is one of the key ideas that this thesis is built on.

There are many more people that have not been listed. I sincerely thank all the people who has helped me. I love you all.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Reinforcement-Learning Approaches	5
1.2 Natural Language Processing	12
1.3 Dog-Inspired Learning	15
1.4 Contributions	17
<b>2 Task Representation</b>	<b>19</b>
2.1 Introduction	19
2.2 Markov Decision Processes	19
2.3 Linear Temporal Logic	20
2.4 "Short-Circuit" Geometric Linear Temporal Logic	22
2.5 Samples	25
2.6 Conclusion	27
<b>3 Interactive Learning through Evaluative Human Feedback</b>	<b>29</b>
3.1 Human-Centered Reinforcement Learning	29
3.2 Human Feedback is Policy-Dependent	30
3.2.1 Empirical Results	32
3.3 Convergent Actor-Critic by Humans	34
3.3.1 Real-time COACH	36
3.4 Comparison of COACH, Q Learning, and TAMER in Simulated Grid World	37
3.4.1 Learning Algorithms and Feedback Strategies	38
3.4.2 Results	39
3.5 Robot Case Study	43
3.5.1 Results	53
3.6 Conclusion	54
<b>4 Teaching Complex Tasks through Decomposition</b>	<b>55</b>
4.1 Training Agent like a Dog	55
4.2 GLTL algorithm	56
4.3 Experiments and results	61
4.3.1 Mission Decomposition Study	61
4.3.2 Mission Decomposition Study results	62
4.3.3 Recomposition Study	63
4.3.4 Recomposition Study results	64
4.3.5 Simulation study	65
4.3.6 User Study: Learning Basic Tasks from Non-expert Human Trainers	67
4.3.7 User study Results: Learning Basic Tasks from Non-expert Human Trainers	68

4.3.8	User study: Learning Complex Missions via Decomposition . . . . .	70
4.3.9	Results of User study: Learning Complex Missions via Decomposition	71
4.4	Conclusion . . . . .	72
<b>5</b>	<b>Interactive Learning from Human Feedback and Natural Language</b>	<b>75</b>
5.1	Why Natural Language? . . . . .	75
5.2	Background . . . . .	75
5.3	Sequence to Sequence Model . . . . .	76
5.3.1	Recurrent Neural Networks . . . . .	77
5.3.2	Long Short-Term Memory . . . . .	78
5.4	Learning Interactively from Natural Guidance ( <i>LING</i> ) . . . . .	79
5.4.1	Attention Mechanism . . . . .	80
5.4.2	Beam Search . . . . .	82
5.4.3	Algorithms . . . . .	83
5.5	Experiment . . . . .	86
<b>6</b>	<b>Conclusions and Future Directions</b>	<b>95</b>
6.1	Humans . . . . .	95
6.2	Tasks . . . . .	96
6.3	Learning from interaction . . . . .	96
6.3.1	COACH . . . . .	96
6.3.2	GLTL-based algorithm . . . . .	97
6.3.3	Similarity score . . . . .	97
6.3.4	LING . . . . .	97
6.4	Natural Guidance . . . . .	98
6.5	Future Directions . . . . .	98
	<b>Bibliography</b>	<b>101</b>

# List of Figures

1.1	Protocol droid C-3PO, and astromech droids BB-8, and R2-D2. . . . .	1
1.2	AI AlphaGo from DeepMind beat Ke Jie, a Go world champion, in a three-game match in May 2017. . . . .	2
1.3	The robots are making a Tesla Model 3 and Model Y at their factory in Fremont, California. . . . .	3
1.4	Autonomous driving from Level 0 to Level 5 according to the Code of the District Columbia. Pictures from <a href="http://www.steinlaw.com">www.steinlaw.com</a> . . . . .	4
1.5	The high-level structure of the interaction in reinforcement learning. . . . .	5
1.6	The high-level structure of the Learning from Demonstration pipeline. . . . .	7
1.7	McQueen. A yorkie who accompanied the author during his entire Ph.D. period. McQueen contributed to this dissertation as an inspiration. The picture shows McQueen before (upper half) and after (bottom half) training. "Yay! I got a treat!" . . . . .	16
2.1	The simple 1 by 3 grid world. . . . .	25
2.2	The change of status of the formula eventually (A and eventually B) at position 0 where $A = \text{True}$ , $B = \text{False}$ . . . . .	26
2.3	The transitions between states of the formula eventually (A and eventually B) in the simple 1 by 3 grid world. . . . .	27
3.1	The goal of the task is to teach the robot to move to the red flag at the bottom left corner. The human trainer gives big positive reward when the robot moves closer to the flag, and small negative reward when it moves away, which leads to a positive reward cycle. . . . .	31
3.2	The training interface that was shown to the participants during the online study. . . . .	32
3.3	The three conditions of the dog in three consecutive episodes. . . . .	33
3.4	The feedback distribution for first step of the final episode for each condition. Feedback tended to be positive for improving behavior, but negative otherwise. . . . .	34
3.5	The simulated $8 \times 5$ grid in which the agent starts in $0,0$ and must get to $7,0$ , which yields $+5$ reward. However, from $1,0$ to $6,0$ are cells the agent needs to avoid, which yield $-1$ reward. . . . .	37
3.6	Task feedback. COACH is without eligibility traces. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale. . . . .	39
3.7	Task feedback. COACH is with eligibility traces. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale. . . . .	40
3.8	Action feedback. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale. . . . .	41
3.9	Improvement feedback. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale. . . . .	42
3.10	The TurtleBot experiment in the Rlab of Brown University. The human trainer is about to train the TurtleBot to learn to "hide" when it sees the pink ball. . . . .	43

3.11	In the training period of the "hide" task, the agent received a punishment signal, which was a negative feedback when it went near the pink ball. The red circle at the upper left shows the reward signal. . . . .	44
3.12	In the "hide" task training, the agent received a reward signal, which was a positive feedback when it went away from the pink ball. The blue square at the upper left shows the reward signal. . . . .	45
3.13	In the verification test of the "hide" task, the agent captured the pink ball and was about to behave according to the policy it had learned. No feedback was given by the human trainer during verification. . . . .	46
3.14	In the verification test of the "hide" task, the agent saw the pink ball and went away from it, which was what it had been trained by the human trainer to do to "hide". No feedback would be given by the human trainer during verification. . . . .	47
3.15	In the "ball following" task, the human trainer gave a positive feedback when the TurtleBot went to the pink ball. . . . .	48
3.16	In the "alternate" task, the TurtleBot was trained to go back and force between the pink ball and a cylinder. . . . .	49
3.17	In the "cylinder navigation" task, the human trainer used the pink ball as a lure to teach the agent to go to the cylinder. . . . .	50
3.18	In the "cylinder navigation" task, the TurtleBot followed the pink ball as the lure and went towards the cylinder. . . . .	51
3.19	In the verification test of the "cylinder navigation" task, the TurtleBot could only see the cylinder. The pink ball lure did not appear in its scene. . . . .	52
3.20	In the verification test of the "cylinder navigation" task, the TurtleBot navigated to the cylinder without seeing the pink ball lure. . . . .	53
4.1	Templates used in constructing temporal formulas. . . . .	59
4.2	The training procedure for task $\square\diamond(\text{table} \wedge \diamond\text{fridge})$ . Steps 1) and 2) use template 1 from Figure 4.1. Step 3) uses template 13. The final step, 4) uses template 2. . . . .	60
4.3	Floor plan images to help ground the questions. . . . .	62
4.4	Experimental domain for teaching temporal tasks. . . . .	65
4.5	The number of training feedback needed for our algorithm, TAMER, and COACH across tasks: $\square\neg\text{chair}$ ; $\diamond\text{table}$ ; $\neg\text{table} \cup \text{charger}$ ; and $\square\text{fridge}$ . . . . .	66
4.6	Task success rate in user study: Learning Basic Tasks from Non-expert Human Trainers. . . . .	69
4.7	Mission success rate in user study: Learning Complex Missions via Decomposition. . . . .	71
4.8	Task success rate in user study: Learning Complex Missions via Decomposition. . . . .	71
5.1	Sequence to Sequence Model. . . . .	76
5.2	Recurrent Neural Networks have internal loops. . . . .	77
5.3	The structure of LSTM unit. The cell state $C_{t-1}$ is the memory from last LSTM unit. The hidden state $h_{t-1}$ is the output of the last LSTM unit. . . . .	78
5.4	The high-level structure of encoder, decoder, and the connection between them. . . . .	79
5.5	Detailed visualisation of how encoder-decoder works. Each time step an input unit is taken by the BiLSTM, it updates its hidden state based on its inputs and previous inputs it has seen. The last hidden state of the encoder is the context passed along to the decoder. . . . .	80
5.6	Structure of the attention mechanism. Attention scores are computed by all the hidden vectors of the encoder and the current hidden vector. . . . .	81

5.7	Encoder-decoder structure with attention. The encoder passes all the hidden states to the decoder. An attention decoder calculates the attention score based on the all the hidden vectors of the encoder and the current hidden vector. . .	81
5.8	Given the input sentence, with Beam width set to 3, the model finds the top 3 words with the highest probability. In this example, the outputs are: "eventually", "always", "and". . . . .	82
5.9	Given the input sentence, with Beam width set to 3, the model will then find the three best pairs for the first and second words based on conditional probability. . . . .	82
5.10	Learning Interactively from Natural Language Model ( <i>LING</i> ). . . . .	84
5.11	The experiment space for non-expert user study. The corners of the virtual grids are marked with small white paper cards on the floor. . . . .	86
5.12	The experiment space for non-expert user study. The 4 by 5 grid world is marked in green virtual lines. There are five atomic propositions in the environment that are known by both the human trainer and the agent: The charger, the fridge, the oven, the window, and the "table" which is actually a kitchen island. . . . .	87
5.13	The photo of Kuri, a robot developed by Mayfield Robotics. . . . .	88





# List of Tables

2.1	Transition $(s, a, s')$ in $MDP_{\diamond_{\mu}\phi_1}$ constructed from a transition $(s_1, a_1, s'_1)$ in $M_{\phi_1}$ . As above, $p(s' s'_1) = \frac{T(s,a,s')}{T_1(s_1,a_1,s'_1)}$ . . . . .	23
-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----



## Chapter 1

# Introduction

*“You’re just a droid. You don’t know about real duty, about what it’s like to have a brother.”  
“I am sorry. I always wanted to have human feelings, but I do not.”*

- Clone trooper Fives and AZI-3  
Star Wars: The Clone Wars

Robots, less commonly known as droids, automatons, are mechanical beings that possessed Artificial Intelligence (AI). They are used in a variety of environments to play multiple different roles, which are often considered menial, repetitive, or risky for other species like human beings. Robots are often designed to work in fields that require extensive specialization and knowledge.



FIGURE 1.1: Protocol droid C-3PO, and astromech droids BB-8, and R2-D2.

In the movies in the "Star Wars" series, which were well known and popular during the period the author was preparing this dissertation, three robots, C-3PO, BB-8, and R2-D2 [1.1](#) contributed a lot to the stories. In the movie, these robots could understand human natural language very well and execute the commands that they received, some of which were high-level, vague, and lacking details. Most of the time, they could even go beyond the given tasks to achieve self-generated goals. In reality, however, it is still very challenging for a robot to interpret open-ended commands from human beings and execute the commands correctly.

Machine Learning (ML) techniques have achieved great success in lots of robotics applications. Nowadays, robots driven by state-of-the-art AI can play the game of Go better than top-level professional human players (Figure [1.2](#)), build (Figure [1.3](#)) and drive (Figure [1.4](#)) our cars autonomously (happening soon from the author's point of view), and vacuum floors by themselves. The algorithms that drive them, however, can handle only very limited complexity in the real world. It is so far impossible to find a robot that you can buy, bring home, and ask to clean the kitchen, feed the dog in a proper way, and not cause more troubles while getting the jobs done. These daily tasks, though seemingly simple and straightforward, are actually ambiguous and possess uncertainties and complexities that ML systems today cannot handle.

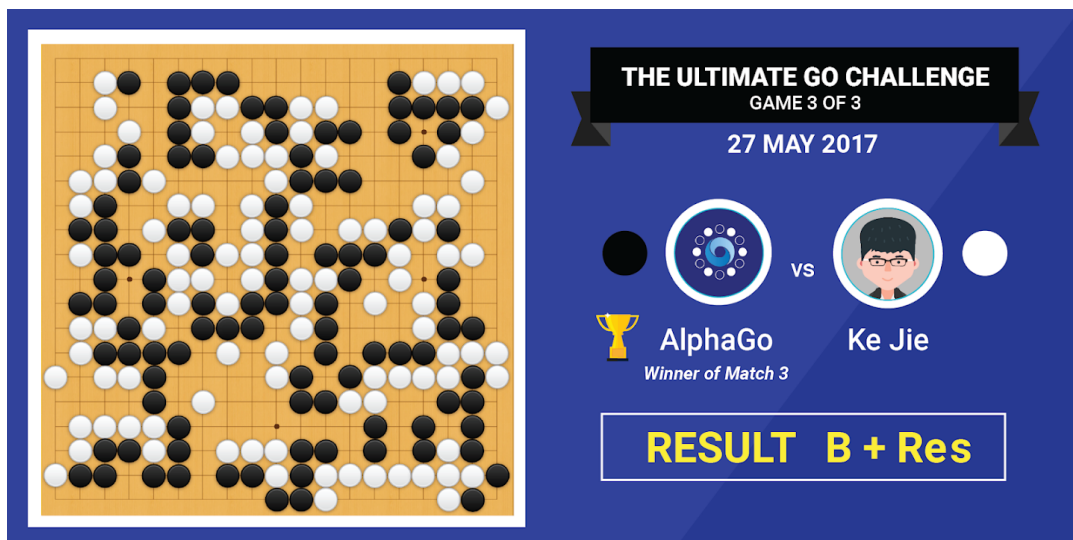


FIGURE 1.2: AI AlphaGo from DeepMind beat Ke Jie, a Go world champion, in a three-game match in May 2017.

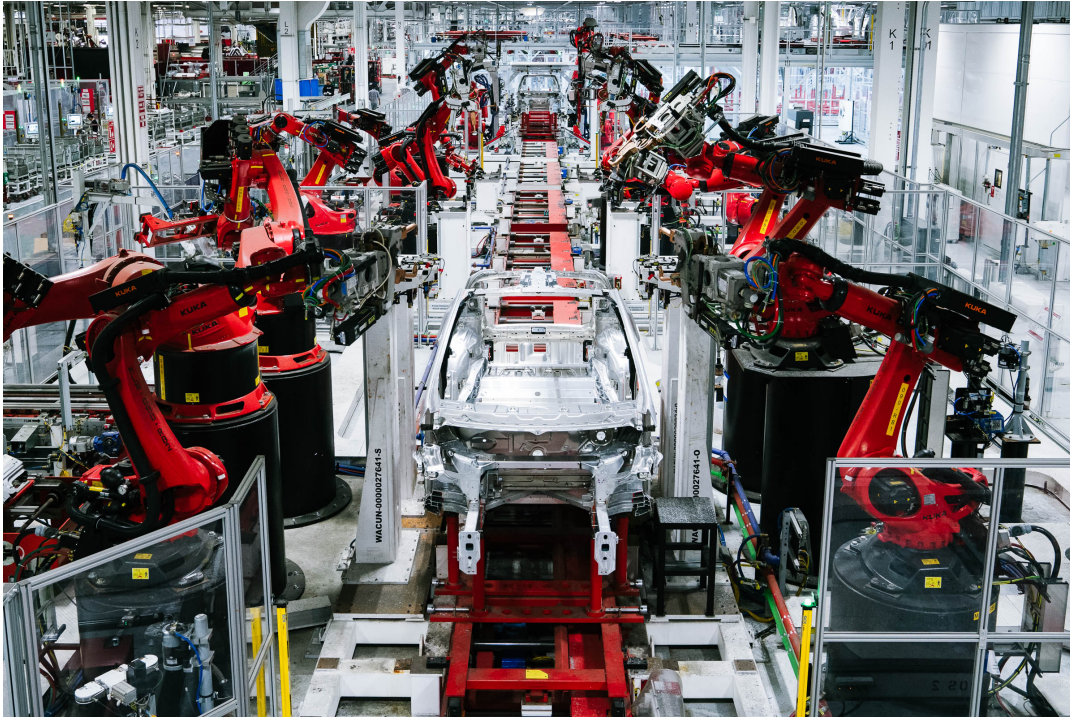


FIGURE 1.3: The robots are making a Tesla Model 3 and Model Y at their factory in Fremont, California.

For on-road vehicles






















		 Human driver	 Automated system		
		Steering and acceleration/ deceleration	Monitoring of driving environment	Fallback when automation fails	Automated system is in control
Human driver monitors the road	<b>0</b> NO AUTOMATION				N/A
	<b>1</b> DRIVER ASSISTANCE				SOME DRIVING MODES
	<b>2</b> PARTIAL AUTOMATION				SOME DRIVING MODES
Automated driving system monitors the road	<b>3</b> CONDITIONAL AUTOMATION				SOME DRIVING MODES
	<b>4</b> HIGH AUTOMATION				SOME DRIVING MODES
	<b>5</b> FULL AUTOMATION				

FIGURE 1.4: Autonomous driving from Level 0 to Level 5 according to the Code of the District Columbia. Pictures from [www.steinlaw.com](http://www.steinlaw.com).

Moreover, different people may have their own preferred ways to control their robots. "Go to the kitchen", for example, could mean "coming to the kitchen immediately!" to the dad, could mean "Follow me to the kitchen without interrupting the others" to the mom, and "Stop touching my legos on the floor and get back to the charger in the kitchen quietly" to the son. The meaning of a command may easily have multiple translations in different environments, and vary according to the person who gives the command.

Last but not least, a robot is useless if the cost of training it is too high. Programming the robot may solve many problems, but it's too difficult for almost all non-experts. Using a remote control seems okay to most people, however, it requires the controller to be able to watch the environment of the robot in real time regardless of the level of complexity of the tasks. Teaching the robot like teaching a dog, if possible, sounds much better, in that it enables the robot to understand and execute tasks by itself, and the cost of training a dog, which could be as simple as providing treats when it performs well and giving alarm signals if it does something wrong, is affordable. Speaking to the robot is much more useful and preferred if the desired behavior is learnable by the robot. In a word, a "smart" robot that is able to understand and learn from human trainers fast will be welcomed by most people easily, a silly one won't be. Therefore, making it possible for non-experts to teach the robot to understand their own communication preferences in an easy, affordable, and reasonably fast way could produce significant value. It's also a fascinating scientific challenge.

## 1.1 Reinforcement-Learning Approaches

Some robots run pre-programmed code to perform the same tasks over and over. Such robots are not “smart” in the way addressed in this dissertation. Some robots rely on rule-based systems and decision trees to finish more complicated tasks. These techniques can be considered AI (artificial intelligence), but such a machine is not capable of learning, and would also not be considered smart. The key to making a robot smart is machine learning. An ML-based robot is able to learn new knowledge through training, and adapt to serve up the most appropriate behavior.

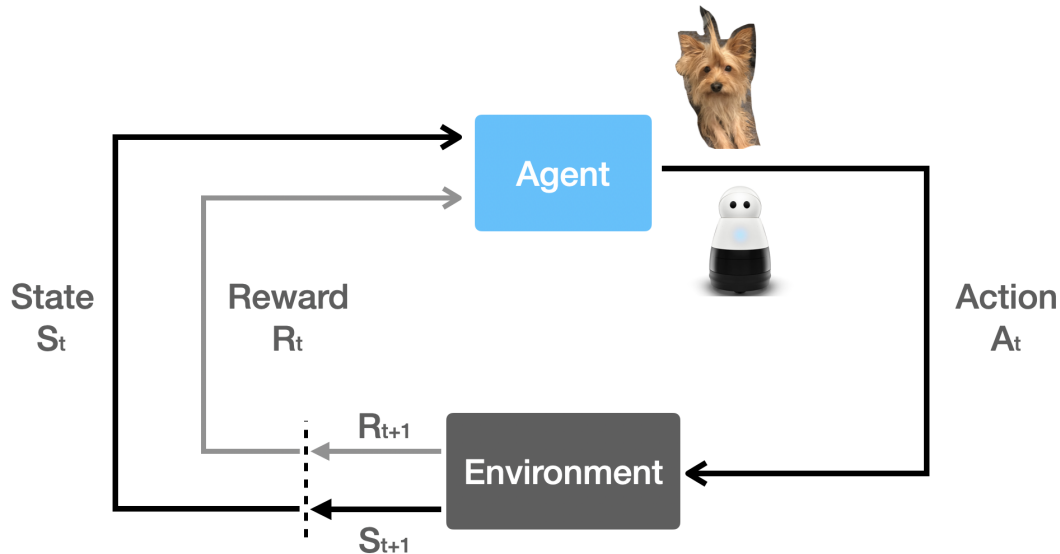


FIGURE 1.5: The high-level structure of the interaction in reinforcement learning.

Reinforcement learning (RL), one of the most important area and the core of machine learning, is a feasible solution to learning in a smart way. The high-level structure (Sutton and Barto, 1998a) of the interaction in RL is shown in Figure 1.5. An RL problem normally consists of:

- **Agent.**  
The learner and decision maker. In robotics, it’s the robot and its software. In real-world dog training scenarios, the dog (depicted by Unicorn McQueen in Figure 1.5) can be considered to be the agent though human beings have an incomplete understanding of their internal learning infrastructure.
- **Environment.**  
The environment contains all information related to the problem so that the agent can learn and decide what actions to perform in what circumstances. In some cases, the trainers know all the information in the environment, in other cases they do not.
- **Action.**  
The set of actions that the agent can choose to perform in the environment. For the robot in Figure 1.5, the actions include the set {stay, move forward, move backward, rotate}.
- **State.**  
The state, or status of the agent in the environment.

- **Reward.**  
It's usually a scalar value in traditional RL formulations. The environment produces a reward for each action selected and performed by the agent.
- **Policy.**  
The control strategy, the decision-making function of the agent. In most of cases it is a mapping function from situations or states to actions.

In this thesis, we'll also talk about a few other important parts of reinforcement learning:

- **Markov Decision Process (MDP).**  
The Markov Decision Process is a probabilistic model of a sequential decision making problem, where state information can be perceived exactly, and the current state and action selected determines a probability distribution on future states. Essentially, only the current action and state information decide the outcome of applying an action to a state, not the preceding actions or states.
- **Value Function.**  
The value function maps from states to real numbers. The number (value) of a state represents the long-term reward achieved starting from the state, and executing a particular policy.
- **Model.**  
The model maps state-action pairs to probability distributions over states. A (transition) model is the agent's view of the dynamics of the environment. In reinforcement learning, not every problem requires the agent to use a model of its environment explicitly.
- **Function Approximator.**  
Function approximation refers to the problem of inducing a function from training examples. Standard function approximators include decision trees, neural networks, nearest-neighbor methods, and a few other approaches.
- **Discount Factor.**  
The discount factor essentially determines how much the RL agent cares about rewards in the distant future relative to those in the immediate future. If the discount factor is equal to 0, the agent will be completely myopic and only learn about actions that produce an immediate reward. On the other hand, when the discount factor is set to 1, the immediate feedback and the long-term reward have same weight to the agent.
- **Temporal Difference (TD).**  
Temporal Difference methods compare temporally successive predictions, and are fundamental to many RL approaches.

In reinforcement learning, an agent, in this problem it is usually the robot, interacts with the environment running a policy. At each state of the environment, it takes an action suggested by the policy, receives a reward, which could be a neutral zero value, and transitions to a new state. For most of the RL approaches, the goal is to learn an optimal policy that maximizes the long-term cumulative reward. For the problem of enabling non-expert human trainers to teach robots, RL agents can learn complex behaviors from signals given by people. Dog training in the real world, for example, is an existence proof that it is possible for people to teach complex tasks using simple positive and negative feedback, like giving the dog a treat when it performs well, or punishment otherwise. We have seen animals successfully trained to provide professional services and solve complex, multi-stage puzzles.

In reinforcement learning, the topics of interactive learning with human in the loop are discussed broadly in the area of Learning from Demonstration (LfD) (Chernova and Thomaz,



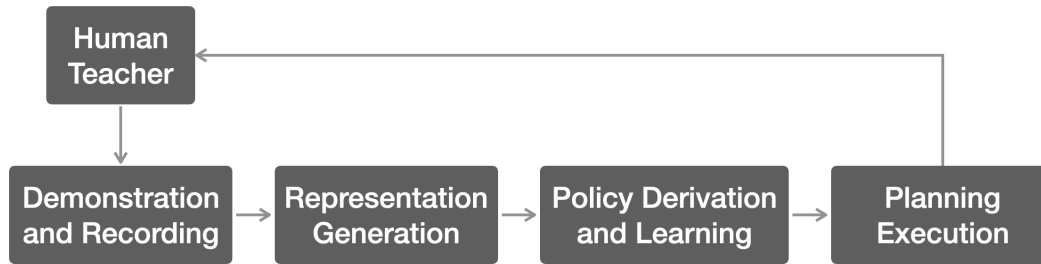


FIGURE 1.6: The high-level structure of the Learning from Demonstration pipeline.

2014), which studies the scenarios where a machine learns the task policy from examples given by a human teacher. The field of LfD has developed extensively over the past three decades; it is also known as Robot Programming by Demonstration (PbD) (Billard et al., 2008), Imitation Learning (Hussein et al., 2017), and Apprenticeship Learning (Abbeel and Ng, 2005). A simplified illustration of the Learning from Demonstration is presented in Figure 1.6. The assumption is that there exists a human teacher who demonstrates execution of an expected behavior. The learner receives the demonstrations and learns to derive a policy from them.

Existing works in the domain structure the learning problem to have the following modules:

1. Data collection.

Data collection is the key and probably most challenging part to a functional supervised learning process. Data collected from multiple sources are usually available in unorganized formats, which means the useful information may be partially observable to ML algorithms as well as being intertwined with random noise. In an RL problem, labeling data in an appropriate way requires the labels to capture the key information of states and actions, and maybe also rewards that the agent needs for future decision making. The diversity, quantity, and quality of the data will determine the training speed, learning curve, performance of the algorithm, and accuracy of the output. In most of cases, manually labeling data for RL problems is very expensive and time consuming. In some cases where human behavior is in the loop, labeling can be impossible.

2. Feature engineering.

In a lot of scenarios, the raw data cannot be used directly by the learning model. It is significant for ML algorithms to decide what input features should be, what similarity metrics look like; in another word, they need to reduce the *dimension* of the data (Fard, Hamzeh, and Hashemi, 2013). The meanings of the features remain intact while the feature space is optimally reduced according to a certain assessment criterion. Information-rich, discriminatory input features should be included so that the algorithm can learn faster. Less important, information-poor, redundant or non-discriminatory features should be excluded so that the algorithm is not misled by useless data. For the majority of machine-learning projects in both academia and industry, feature engineering takes more time than all the other processes.

3. Reward function definition.

The reward function plays a key role in reinforcement learning. Reward engineering defines the objective of a learning algorithm. A lot of traditional RL environments, handled by algorithms such as Q-Learning (Watkins, 1989), include goal reward—the only reward that the agent receives is when the task is finished. In some problems like the game of Go, there is only one reward for all the actions, and it is delivered

when the game is over. In other scenarios like some Atari games, some reward (points) are provided on each step. Human feedback as reward, on the other hand, is more complicated. Human feedback is often ambiguous, and may be wrong for the goal of the task. It is hard to consider people as ideal trainers. Dog training courses, for example, are usually designed to teach not the dogs but the dog owners, who will teach their dogs during daily training, the proper way of giving feedback. A learning algorithm needs to be able to learn from such feedback. Algorithms such as TAMER (Knox and Stone, 2009a) were designed for such problems. The other main problem about learning with human-in-the-loop is the learning speed. Traditional RL algorithms often requires many repetitions for the agent to explore with trial and error before mastering their task. Therefore, they are usually very slow. When the rewards come from real human trainer, such a procedure can easily be too long and boring. Therefore, reward-function engineering, or reward shaping, was developed to improve the agent's learning performance in complex tasks. Reward shaping works by adding a supplemental reward, often in the form of a potential function, to the environmental reward. The potential function is usually predefined by the agent designer, though it also can be learned. In this thesis, we examine human feedback in Chapter 3 and introduce algorithms and experiments to solve the problem.

#### 4. Task design.

Teaching a dog complicated missions, for example, to fetch a frisbee, bring it back, then sit down and wait for the next command, is hard for most of new dog owners. However, most dogs can learn a part of it, sit down, when there's positive reward or treats. If the mission can be decomposed into several less complicated sub-tasks, and the sub-tasks can be finished in parallel, the learning procedure will speed up. On the other hand, a mission may be too complicated to learn by the RL algorithm directly, but if it can be split into a set of simple sub-tasks, learning may be possible. How to design the task is an important part of RL and LfD. Chapter 2 presents additional details.

Human-centered reinforcement learning has been developed and proven to be a powerful method for facilitating ordinary people to teach agents in a more natural way. Similar to standard reinforcement learning, the core problem of human-centered RL is designed to solve the challenge of human-delivered feedback. Most interactive RL algorithms are distinguished by interpretations of human feedback (Amershi et al., 2014). Based on existing interpretations, there are three major types of human-centered RL algorithms developed that facilitate agents to learn from them (Li et al., 2019):

- Interactive shaping, also known as "Learning from human reward".  
Similar to traditional reinforcement learning, in interactive shaping, agents learn from human feedback in a representation of numeric reward values.
- Learning from categorical feedback.  
Where the agents interpret human feedback as categorical feedback strategies that depend on not only the desired behavior that the teacher is trying to teach, but also the teacher's training strategy.
- Learning from policy feedback.  
For learning from policy feedback, the agents formalize the meaning of human feedback as a comment on the agent's behavior based on the expected agent policy or the policy the agent is following, and use it directly as policy feedback.

In interactive shaping, a human observer provides shaping rewards to train an agent to perform a task. Isbell et al. (2006) used both reward and punishment to train an artificial agent,

Cobot, where they applied reinforcement learning in an online text-based virtual chatting scenario. Since then, a set of similar animal-training methods have been applied, including clicker training.

Value-function methods use temporal difference (TD) to estimate the expected return. The value function reports the expected utility of taking a given action in a given state. The policy can be derived from the learned value function straightforwardly. The value function is equivalent to the reward function when the learning is set to be myopic.

Algorithms for learning from human reward can be briefly categorized based on whether the effect of an agent’s action on future states is taken into consideration or not (Li et al., 2019):

- Learning from myopic human reward with value functions.  
The existing works that inspired this thesis include: Knox and Stone (2009a).
- Learning from myopic human reward with actor-critic methods.  
The existing works that inspired this thesis include: Ngo Anh Vien and Ertel (2012).
- Learning from nonmyopic human reward with value functions.  
The existing works that inspired this thesis include: Thomaz and Breazeal (2008), Tenorio-Gonzalez, Morales, and Villaseñor-Pineda (2010), Thomaz and Breazeal (2006), Isbell et al. (2006), Suay and Chernova (2011), León et al. (2011).
- Learning from nonmyopic human reward with actor-critic methods.  
The existing works that inspired this thesis include: Pilarski et al. (2011).

A common class of RL algorithms is *actor-critic* algorithms. Bhatnagar et al. (2009) provide a general template for these algorithms. Actor-critic algorithms are named for the two main components of the algorithms:

- The actor.  
A parameterized policy that dictates how the agent selects actions.
- The critic.  
The critic estimates the value function for the actor and provides critiques at each time step that are used to update the policy parameters.

Typically, the critique is the TD error:

$$\delta_t = r_t + \gamma V(s_t) - V(s_{t-1}),$$

which describes how much better or worse a transition went than expected. Actor-critic approaches inspired the work in this dissertation, and are discussed further in Chapter 3.

In the work of Training an Agent Manually via Evaluative Reinforcement (TAMER) (Knox and Stone, 2009a), a framework that learns myopically by directly modeling human reward was produced. TAMER considers an agent learning in the framework of MDP without a reward function. TAMER was the state-of-the-art in the domain when the work in Chapter 3 was conducted, and inspired some of the key ideas in this dissertation. In TAMER, a human teacher observes the agent’s behavior and gives rewards based on the evaluation of its quality. The agent learns a function,

$$R_H(s, a) = \omega \phi(s, a),$$

a parameterized function that approximates the expectation of experienced human rewards,

$$R_H : S \times A \rightarrow R,$$

where

$$\omega = (\omega_0, \omega_1, \dots, \omega_{m-1})^T$$

is a column parameter vector, and

$$\phi(x) = (\phi_0(x), \phi_1(x), \dots, \phi_{m-1}(x))^T$$

with  $\phi_i(x)$  as the basis function,  $i = 0, 1, \dots, m - 1$ , and  $m$  is the total number of parameters. Given a state  $s$ , the action with the largest estimated expected reward is picked.

TAMER includes three key components:

1. Credit assignment.
2. A predictive model of reward from the human trainer.
3. An action-selection function with the learned model of reward from human trainer, for example, greedy, greedy epsilon.

In reality, human trainers watch the agent's behavior, think for a short while, and send the feedback. The duration of such thinking varies. As a result, the agent is uncertain which time step the reward is associated with. TAMER uses a credit-assignment method to deal with this delay. A probability density function for estimating the probability of the teacher's feedback delay is introduced, which provides the probability that the feedback is given within any specific time interval. The agent uses the function to compute the probability that a single reward signal should be assigned to the target time step.

For example, a probability density function  $f(t)$  is used to define the delay of the reward generated by the trainer. At time step  $t$ , the credit for each previous time step  $t - k$  is:

$$c_{t-k} = \int_{t-k-1}^{t-k} f(x) dx.$$

If there are multiple rewards within one time step,  $h$  is used to sum all the rewards. The TAMER agent takes  $h$  and the state-action pair as a supervised-learning sample to learn the reward function  $R_H(s, a)$ . At time step  $t$ , the TD error  $\delta_t$  is

$$\delta_t = h - R_H(s, a),$$

where

$$R_H(s, a) = \omega \phi(s_t, a_t).$$

Based on the gradient of the squared error, the parameter of  $R_H(s, a)$  is updated with incremental gradient descent as

$$\omega_{t+1} = \omega_t - \alpha \nabla_{\omega} \frac{1}{2} \{h - R_H(s_t, a_t)\}^2 = \omega_t + \alpha \delta_t \phi(s_t, a_t),$$

where  $\alpha$  is the learning rate.

The original TAMER framework only works in domains with discrete action space, but Vien, Ertel, and Chung (2013) extended the TAMER framework to continuous state and action spaces by proposing actor-critic TAMER. Actor-critic TAMER considers the policy as the actor, and the function of reward from human trainer as the critic. The reward function,

$$R_H(s) = \omega^T \phi_r(s),$$

depends on state information. The policy is represented explicitly as

$$\pi(s, a) = \theta^T \phi_{\pi}(s, a).$$

The actor chooses actions at given states and the critic is used to evaluate the performance of the actor. The critic’s evaluation provides a TD error  $\delta_t$  as the gradient estimate of a specific performance measure to improve the actor by updating its parameters. The difference is that the policy  $\pi(s, a)$  is updated by

$$\theta_{t+1} = \theta_t + \beta \delta_t \phi_{\pi}(s_t, a_t),$$

where  $\beta$  is the learning rate for the policy.

Knox and Stone (2009a) introduced VI-TAMER, a model-based RL method, which supports the agent when learning from nonmyopic human reward. In VI-TAMER, an agent learns from discounted human reward and models the human reward at the same time. Pilarski et al. (2011) introduced a continuous model-free RL algorithm that learns an optimal control policy using only human reward via actor-critic. The algorithm learns from discounted human reward, and works in continuous state and action spaces. Unlike VI-TAMER, their algorithm does not model human reward signals explicitly. Instead, they treat human rewards in the same way as environmental rewards in traditional RL.

Warnell et al. (2017) extended the TAMER framework with Deep Neural Networks (DNNs) to learn complex tasks. It was able to work in high-dimensional state spaces. They evaluated Deep TAMER on the challenging Atari game of BOWLING with pixel-level state features, and it turned out that agents trained by humans using Deep TAMER significantly outperformed agents trained by humans using TAMER. Moreover, Deep TAMER agents were able to achieve higher scores than agents trained using state-of-the-art deep reinforcement-learning techniques.

TAMER was also tried with the agent learning solely from facial expressions (Li et al., 2020). The results showed that bi-directional feedback and competition could improve the accuracy of estimated feedback and with appropriate prediction models, facial expression would significantly improve the performance of agents.

Closely related to this thesis, Loftin et al. (2016) assume that human teachers provide feedback with different strategies, and for strategy-aware learners, neutral feedback also contains information from the trainer. Human feedback is grouped into four categories: positive reward, negative reward, positive punishment, and negative punishment. Reward is considered as a stimulus to increase the frequency of an associated behavior, punishment moves things in the other direction. Positive reward enhances the stimulus, and negative reward weakens the stimulus. Such a scheme makes it possible for different human trainers to employ a different feedback strategy to the same behavior of the learner. The authors introduced Strategy-Aware Bayesian Learning (SABL) and Inferring Strategy-Aware Bayesian Learning (I-SABL) to learn with less discrete feedback than existing techniques, while taking as few exploratory actions as possible.

Other works in related domains also inspired this thesis besides TAMER and SABL:

Christiano et al. (2017) proposed an approach which explored goals defined in terms of human preferences between pairs of trajectory segments. They claimed that their method can effectively solve complex RL tasks without access to the reward function and therefore reduce the cost of human oversight.

As identified in Chapter 3, inconsistent human feedback is a bottleneck to achieving the best performance, Ramesh et al. (2020) experimentally verified that human trainers subjected to perceptual contrast effects underrate or overrate an agent’s actions when previously exposed to an agent with higher or lower competence on the same task. Furthermore, they showed that not accounting for these effects when incorporating human feedback in on-policy reinforcement-learning methods leads to deleterious outcomes in agent-training procedures. Ghadirzadeh et al. (2020) presented a reinforcement-learning-based framework for human-centered collaborative

systems. The framework can not only effectively deal with the uncertainties in perception, but also find an optimal balance between timely actions and the risk of making mistakes.

Kim et al. (2017) proposed an Intrinsic Interactive Reinforcement Learning (IIRL) method that is an effective way to communicate with the learner. They regarded an error-related potential (ErrP) and an event-related activity in the human electroencephalogram (EEG) as an inherent implicit feedback for RL. The robot learns the meaning of the gesture by learning the assignment of human gestures to the corresponding actions. At the same time, the robot learns the mapping between human gestures and robot actions via a contextual bandit method. The results showed that the higher the performance of online ErrP detection, the smaller the number of errors of the robot for most subjects.

Because of the rapidly increasing solution space, it is hard for a robot to learn an assembly sequence. De Winter et al. (2019) proposed an idea that uses human knowledge to reduce the solution space. The method combined Interactive Reinforcement Learning (IRL) to learn from human advice and Potential-Based Reward Shaping (PBRs) to focus on a smaller part of solution space. Experiments showed that the communication model based on advice constraints, translated from natural language, is more efficient, as it requires the fewest interactions while still converging more quickly than other communication models.

Similarly, the work of Ayala, Henríquez, and Cruz (2019) used IRL to train the agents and tackled the problem of continuous representations along in the interactive approach. It turned out that using continuous states and interactive feedback converges faster compared to discrete and autonomous reinforcement learning respectively.

Tabrez and Hayes (2019) proposed a framework for estimating and improving a collaborator's task comprehension and execution. By characterizing the problem of sub-optimal performance as evidence of a malformed reward function, they introduced mechanisms to both detect the root cause of the sub-optimal behaviour and provide feedback to the human collaborator to repair their decision-making process.

Yu et al. (2020) proposed a general approach that utilized online human demonstrations to directly instruct an agent's behaviors. The method could alleviate the uncertainties caused by human assessments and eliminate the offline pre-training used in most existing learning-from-demonstration approaches. Using the approach, the authors investigated the interplay among different shaping methods between humans and agents. The proposed adaptive shaping algorithm, due to the interplay between different methods, could take the benefits of each method to achieve a more robust and efficient learning performance.

Unlike most IRL approaches that require advice from experts, Celemin et al. (2019) proposed to use human corrective advice in the actions domain for learning motor trajectories. In policy search, they combined human feedback with reward functions. Compared with standard reinforcement learning without human advice, the results showed that the proposed method not only converges to higher rewards when learning movement primitives, but also that the learning is sped up by a factor of 4 to 40 times, depending on the task.

## 1.2 Natural Language Processing

Another main topic of this thesis is Natural Language Understanding (NLU). Interactive robot teaching with evaluative feedback makes it possible for non-expert human trainers to communicate with a robot, but that is still less natural than what human beings would prefer, which is "speaking". NLU techniques enable the natural language that people use to be translated to executable formal language that can be interpreted by machines accurately.

Natural Language Interfaces (NUI) have long been a topic of Human-Robot Interaction (HRI). SHRDLU (Winograd, 1980), for example, was an early natural language understanding program, which allowed human users to carry on a conversation with the computer, naming

collections and querying the state of a simplified "blocks world". The area has grown fast in recent years. Existing work typically view the problem as a single step process, in which a human operator gives an instruction and an automated agent is evaluated on its ability to execute it (MacMahon, Stankiewicz, and Kuipers, 2006; Branavan et al., 2009).

A common way of understanding the meaning of a natural language command is to map the utterance into a formal representation through semantic parsing (Zelle and Mooney, 1996). Due to the complexity of natural language and the space of goals of the command, the size of the logical forms produced may be open ended, therefore, structured prediction of the logical form, for example, tree structures, is widely adopted.

Supervised learning of structured prediction models have been proposed and well discussed during the past two decades (Zettlemoyer and Collins, 2012). Fully annotated logical forms to be used as training samples are often hard to collect due their expense. If the structure is simple, its ability of representing rich information is limited. If the logical forms are complicated, which is common in many cases, only well trained people who are familiar enough with both the natural language and the logical language can manually produce useful language pairs. The tasks become labor-intensive and time-consuming, which limits the expansion of the datasets.

Among most of the work in the area, NLP systems built by ML techniques can be amazingly effective when plentiful labeled training data exists for the task. Unfortunately, for broad coverage (both in task and domain and language) language understanding, it is rarely the case where there is sufficient labeled data, and the system designer must find some other ways for the AI system to learn, either through unsupervised learning over unlabeled data, or through interaction with people (Daumé III, 2009). The current state of the art in different areas of NLP is still very far from allowing fully automatic high quality results, therefore human intervention is required to correct the output of the NLP engines.

Remarkable work has been done trying to solve the problem (Clarke et al., 2010; Liang et al., 2016). Logical-form denotations as supervision enables part of the labeling work to be acceptable by non-expert human labelers via weaker forms of language pairs (Krishnamurthy, Dasigi, and Gardner, 2017; Pasupat and Liang, 2015).

In a dialog or question-answering problem, data labels can be answers to the question posed instead of a logical language translation of the query; this type of labeling can be carried out by a bigger group of non-expert human lablers (Wang, Berant, and Liang, 2015). Zheng et al. (2019) proposed a systematic framework for answering natural language questions through interaction. They let the end users verify the candidate mappings to cope with the ambiguities and presented a query pre-fetching technique. The experiments over three benchmarks showed that their interactive method for answering natural language questions through knowledge graphs is promising and effective.

Branavan et al. (2009) presented an RL approach for mapping natural language instructions to sequences of executable actions. During training, the learner repeatedly constructed action sequences for a set of documents, executed those actions, and observed the resulting reward, which required few or no annotated training examples. Their approach is built on the premise that the reward function that defines the quality of the executed actions should be known.

Thomason et al. (2019) leveraged conversations with humans to expand small, hand-crafted language understanding resources both for translating natural language commands to abstract semantic forms and for grounding those abstractions to physical object properties. They made several key assumptions, and the actions performed by the robot could be decomposed into discrete semantic roles such as patient and source. Their research provides some inspirations for further study.

Another important direction focused on bridging the gap between natural language commands and the physical world used a set of pre-defined templates characterized by a small vocabulary and grammar (Thompson et al., 1993; Mavridis, 2015; Klingspor, Demiris, and

Kaiser, 1997). Scientists have focused on grounding visual attributes and on learning spatial relations and actions for small vocabularies with hard-coded abstract concepts (Roy, 2002; Steels and Vogt, 1997; Matuszek et al., 2014; Kollar, Krishnamurthy, and Strimel, 2013).

In the last decade, Deep Learning (DL) methods for grounding spatial language is getting attention as well (Bisk, Yuret, and Marcu, 2016; Tan and Bansal, 2017; Patel, Pavlick, and Tellex, 2020). Here, human instruction is provided and an outcome is observed. Mehta and Goldwasser (2019) considered the problem as an interactive process, in which the human operator can observe the agent's response to their instruction and adjust it by providing advice consisting of a short sentence as online feedback.

As ground-breaking achievements in applying deep neural networks in other domains occurred, such as computer vision and image processing, we have seen a clear trend in using deep learning to solve classical natural language problems. Mudgal et al. (2018) explored the combination of different choices for deep learning approaches for entity matching with three different problem types and compared the effect of performance improvement with Magellan as a baseline. It was shown that the deep learning solutions outperformed Magellan on textual and dirty data. Overall, they provided comprehensive experiments that paved the path for further investigations.

On the other hand, Bender and Koller (2020) argued that language modeling does not learn meaning, instead, it only learns the form of the data. A clear understanding of the distinction between form and meaning was of great help to select the right hill to climb towards human-analogous natural language understanding. Kassner and Schütze (2020) found that pre-trained language models were extremely poor at handling negated sentences and sentences that included distracting material. When facing question-answer pairs in open domains, these models treated tasks more like shallow pattern matching.

In the specific domain of interactively learning tasks through natural language processing, many classical studies investigated only learning of actions or grounding of words, but not both. Additionally, they often used only a small set of tasks as well as very short and unnaturally simplified utterances. Roesler and Nowé (2019) introduced a framework that used RL to learn actions for several tasks and cross-situational learning to ground actions, object shapes and colors, and prepositions by integrating the above-mentioned two components. The proposed framework was evaluated through a simulated interaction experiment between a human tutor and a robot. The results showed that the employed framework can be used for both action learning and grounding.

Roesler et al. (2019) introduced a probabilistic model for grounding unknown synonymous objects and actions and evaluated it through an interaction experiment between a human tutor and Human Support Robot (HSR). The results showed that both semantic and syntactic information enable grounding of unknown synonyms and that the combination of two achieved the best grounding.

Patki et al. (2019) proposed a model that utilized encoded environmental information within instructions for accurate and efficient natural language understanding. Experimental results on two robots operating in different environments verified that compact environment representations were learned through the content and structure of instructions.

By avoiding loss function construction and increasing the probability of zero-shot obedience to previously unheard commands, scalable human-robot interaction would make progress. Experiments by Matthews and Bongard (2020) showed that a crowd of non-experts can not only discover robots responsive to natural language commands, but also robots that obey similar commands in similar ways.

Ni et al. (2020) proposed two structures: RCNN-BiGRU-CRF and BiGRU-Att-CapsuleNet-BiGRU-CRF, which were used for an Internet Detection and Slot Filling joint task for Internet-of-Things (IoT) Speech Understanding system, which they transferred to the medical field to solve NLU issues in the construction of clinical voice assistants. The hybrid model structures



in this work can also help the semantic understanding module in IoT to realize key functions of NLU in multiple scenarios well.

These approaches reduce the overall data-collection cost, but still require dedicated annotation work to be done before the actual interaction between the final end users and the models, which is not ideal. Resolving this concern inspired the work in this dissertation, and is further discussed in Chapter 6.

### **1.3 Dog-Inspired Learning**

This work introduces an implemented cross-domain approach that allows a non-technical, non-expert, potentially-biased person to teach a robot agent to execute a complicated task with natural human interventions. Specifically, it assumes the trainer:

1. is a general lay person that does not necessarily have a programming background.
2. is a non-expert that will not produce flawless expert demonstrations.
3. cannot give formulated or normalized commands according to some rubrics.
4. oftentimes provides ambiguous interactions that do not reflect his/her actual cognitive state.
5. is impatient, meaning that his/her time is costly.

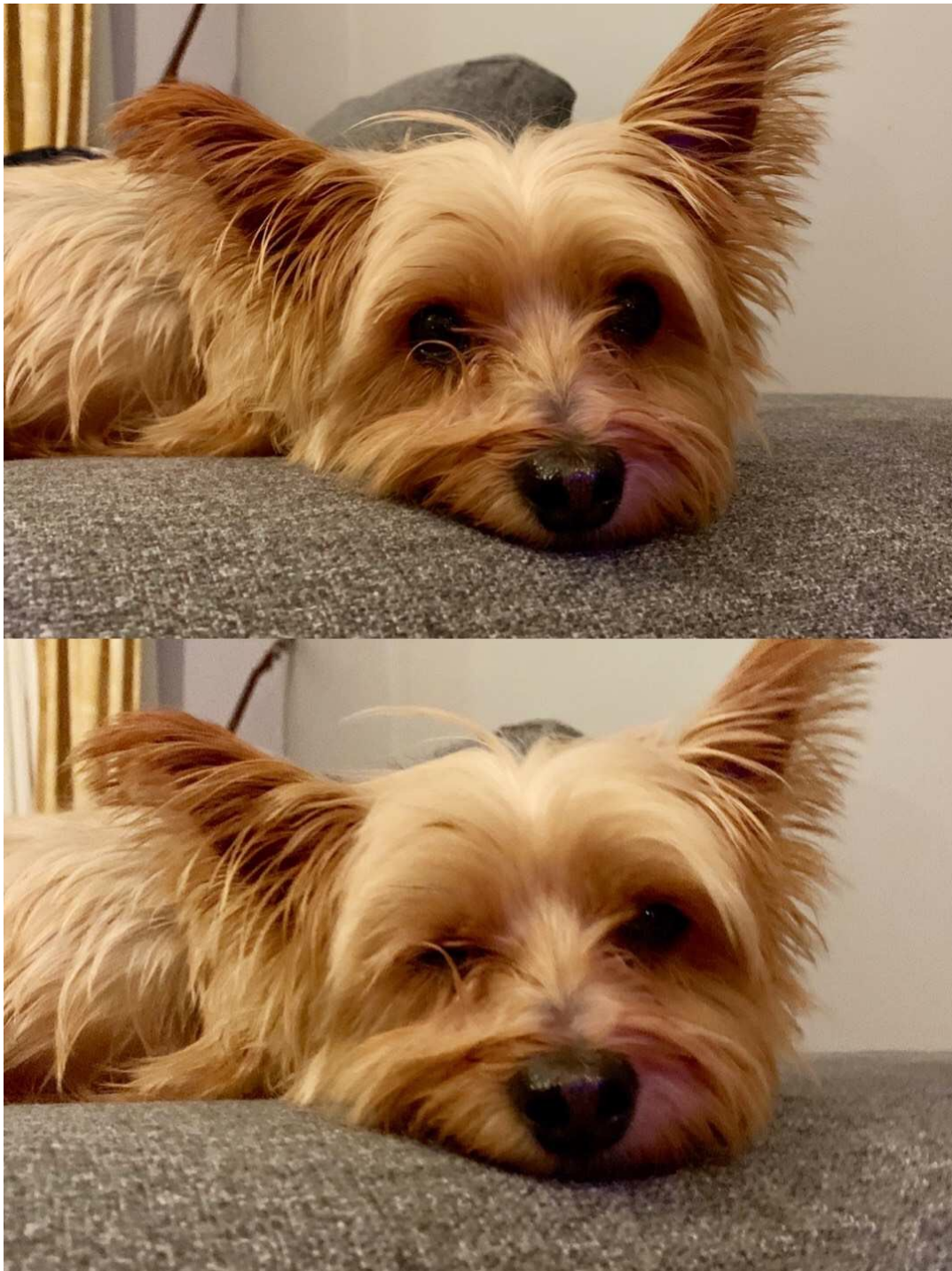


FIGURE 1.7: McQueen. A yorkie who accompanied the author during his entire Ph.D. period. McQueen contributed to this dissertation as an inspiration. The picture shows McQueen before (upper half) and after (bottom half) training. "Yay! I got a treat!"

In accurately capturing the problem and developing an adaptive solution, the author takes inspiration from teaching a dog, specifically one whose name is McQueen (Figure 1.7), to perform desired behaviors following different human language commands. McQueen is a Yorkshire terrier who met the author at the beginning of his Ph.D. journey, and has chosen to stay with him ever since (maybe partially because he kept receiving positive rewards along the way). He has inspired the author to think deeply about how a dog learns and behaves, and what

a trainer should do to teach him in an effective way. Those ideas and experiences contribute to Chapter 2 and Chapter 3 of the dissertation. The approach, though still quite far from the unattainable intelligence of a puppy’s brain, seeks to mimic the process of understanding human natural languages from scratch through interaction and decomposition, without relying on pre-labeled data.

## 1.4 Contributions

This dissertation focuses on the problem of enabling non-expert human trainers to teach complicated tasks to machines through natural modes of interaction.

Chapter 2 introduces Geometric Linear Temporal Logic (GLTL) as a way of representing tasks of arbitrary complexity. It presents detailed instructions of how a GLTL formula can be converted to a Markov Decision Process (MDP), and how to generate its policy through value iteration. A key advantage of the GLTL representation is that a GLTL formula of high complexity can be decomposed into simpler GLTL formulas, and a group of simple GLTL formulas can build up complicated tasks efficiently.

Chapter 3 demonstrates that human feedback is policy-dependent, which means the feedback that a human trainer gives to the agent is determined by the level of improvement of the current action from the learned policy, not the quality of the current action independently. A set of user studies support the claim.

CONvergent Actor-Critic by Humans (COACH) is proposed to learn from such policy-dependent evaluative human feedback. Experiments in simulation environment compare the performance of COACH, Q Learning, and TAMER, where each algorithm was tested with three different types of feedback: sparse goal reward, stationary action reward, and improvement reward, which is close to evaluative human feedback. The results suggest that COACH with eligibility traces learns robustly with all three kinds of feedback, while Q Learning and TAMER could only work with limited feedback types.

A qualitative experiment with a physical robot, TurtleBot, which runs real-time COACH, a special version of COACH, was conducted. The experiment suggests that COACH can scale to a complex domain involving multiple challenges, including fast decision cycle, noisy non-Markov observations from a camera, and hidden agent perception from the trainer. COACH successfully learned all of the five tasks while TAMER failed to learn the compositional behaviors.

Chapter 4 presents a GLTL-based approach to learning tasks of arbitrary complexity through decomposition. It describes the algorithm in detail, and experimental results in a simulation environment. The results show that the GLTL-based algorithm outperforms COACH and TAMER significantly. The feedback it takes for the GLTL-based algorithm to finish four simple tasks is much less than that required by the other two algorithms. For another five complicated tasks that possess compositional structure, GLTL-based algorithm can handle all of them well while both TAMER and COACH failed to learn.

Furthermore, a series of user studies suggest that:

1. Non-expert people can decompose complex tasks into simpler units effectively.
2. The GLTL-based algorithm enables non-expert human trainers to teach simple tasks through evaluative feedback given by the trainers.
3. Complex missions can be taught by non-expert human trainers through decomposing them into simpler sub-tasks, and training the sub-tasks to build up the missions.

In Chapter 5, a deep sequence-to-sequence (Seq2Seq) approach is used to interpret natural language commands and natural language feedback. The Seq2Seq algorithm can learn to

convert natural language to formal language—GLTL formulas—with minimal training data provided incrementally.

Based on the insight of COACH, GLTL, and the Seq2Seq model, I introduce the approach of Learning Interactively from Natural Guidance (LING), which enables the agent to learn tasks of arbitrary complexity through decomposition and a combination of evaluative feedback and natural language.

A qualitative user study suggests that a non-expert human trainer can teach a physical robot running LING to learn complex tasks in a realistic scene, a living room, by giving commands and feedback in natural language.

## Chapter 2

# Task Representation

### 2.1 Introduction

This chapter is about the representation of tasks in reinforcement learning. I introduce the pros and cons of Markov decision processes (MDPs) with reward functions, the traditional way of describing a task in reinforcement learning, then present Linear Temporal Logic (LTL), a task-specification language that has been used for temporal goals, which are not well handled by reward functions. I further explain the limitations of LTL in the scenario of learning a real-world problem, and propose "Short-Circuit" Geometric Linear Temporal Logic, a semantics for LTL that combines the advantages of both LTL and reward functions.

### 2.2 Markov Decision Processes

A variety of scenarios in the area of artificial intelligence (AI) can be formulated as agents making sequential decisions in a discrete or continuous state space. Usually, the goal of an agent is to make decisions optimally, meaning in a way that the long-term cumulative reward that the agent receives in the whole process is maximized. MDPs have been widely used as a standard mathematical framework for modeling such settings.

An MDP consists of five key elements:

$$\langle S, A, T, R, \gamma \rangle,$$

where

- $S$  is the set of possible states of the environment.
- $A$  is the set of actions available to the agent.
- $T(s'|s, a)$  is the transition function, which defines the probability of the environment transitioning to state  $s'$  when the agent takes action  $a$  in environment state  $s$ .
- $R(s, a, s')$  is the reward function specifying the numeric reward the agent receives for taking action  $a$  in state  $s$  and transitioning to state  $s'$ .
- $\gamma \in [0, 1]$  is a discount factor specifying the degree to which distant rewards are preferred to more immediate rewards.

A *stochastic policy*  $\pi$  for an MDP is a per-state action probability distribution that defines an agent's behavior:

$$\pi : S \times A \rightarrow [0, 1],$$

where

$$\sum_{a \in A} \pi(s, a) = 1, \forall s \in S.$$

In the MDP setting, the goal is to find the optimal policy  $\pi^*$ , which maximizes the expected future discounted reward when the agent selects actions in each state according to  $\pi^*$ :

$$\pi^* = \arg \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi\right],$$

where  $r_t$  is the reward received at time  $t$ . Two important concepts in MDPs are:

- the value function ( $V^\pi$ )
- the action–value function ( $Q^\pi$ ).

The value function defines the expected future discounted reward from each state when following some policy

$$\pi : V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi\right]$$

and the action–value function defines the expected future discounted reward when an agent takes some action in some state and then follows some policy  $\pi$  thereafter:

$$Q^\pi(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right].$$

These equations can be recursively defined via the Bellman equation (Bellman, 1957):

$$V^\pi(s) = \sum_a \pi(s, a) Q^\pi(s, a)$$

and

$$Q^\pi(s, a) = \sum_{s'} T(s' | s, a) [R(s, a, s') + \gamma V^\pi(s')].$$

For shorthand, the value functions for the optimal policies are usually denoted  $V^*$  and  $Q^*$ .

In reinforcement learning, an agent interacts with an environment modeled as an MDP, but does not have direct access to the transition function or reward function and instead must learn a policy from environment observations.

MDPs serve as good models in most problems where a Markov assumption is true: At each step, the agent could encode all the information relevant to predicting the effects of actions in its internal representation. However, there are problems, some of which are important, that cannot be formulated as an MDP. Such non-Markovian tasks are also named hidden-state tasks, because some important state information is hidden from the agent’s representation of the current state.

Consider a simple task where the state information consists of the current location of the agent and the environmental information, ‘Please only go to the table if you’ve reached the window’, the conditional probability of the agent to decide the actions of the current step depends on, not only the current state, but also historical information. Therefore, it’s very hard for many traditional RL models to work well with these kinds of tasks.

## 2.3 Linear Temporal Logic

Linear temporal logic (LTL) has been widely used as a specification language for complicated missions (Manna and Pnueli, 1992; Baier and Katoen, 2008). The purpose of temporal logic is the analysis of arguments about events and process in time. LTL’s expressiveness is enriched by a set of atomic propositions and a group of logical operators, for example:

- negation ( $\neg$ ),

- disjunction ( $\vee$ ),
- conjunction ( $\wedge$ ),
- material implication ( $\rightarrow$ ),
- next ( $\bigcirc$ ),
- always ( $\square$ ),
- eventually ( $\diamond$ ),
- until ( $\mathcal{U}$ ),
- release ( $R$ ).

Using these operators, there are a list of fundamental LTL representation formulas:

- $\neg a$ : " $a$  has to be true at the current state",
- $a \vee b$ : "Either  $a$  or  $b$ , one of them has to be true at the current state",
- $a \wedge b$ : "Both  $a$  and  $b$  have to be true at the current state",
- $a \rightarrow b$ : "If  $a$  is true, then  $b$  is true at the current state",
- $\bigcirc a$ : " $a$  has to be true at the next state",
- $\square a$ : " $a$  has to be true from the current state and stay true in all states forever",
- $\diamond a$ : " $a$  has to be true at least one time in the future",
- $a \mathcal{U} b$ : " $a$  has to stay true at least until  $b$  is true, if  $b$  is true then  $a$  does not have to be true any more",
- $a R b$ : " $b$  has to stay true until and including the point where  $a$  becomes true for the first time, if  $a$  never becomes true,  $b$  has to be true forever".

The task ‘Please only go to the table if you’ve reached the window’ can be expressed as

$$(\neg \text{table}) \mathcal{U} \text{window}.$$

More complicated missions can be formed through composition of these fundamental LTL operators (Baier and Katoen, 2008). Once an LTL formula is chosen for a given RL environment, the goal of the agent is to choose behavior that can maximize the probability that the LTL formula expression is satisfied. The main advantage of using LTL is that it can represent missions that are hard for simple reward functions, for example, non-Markov tasks. Most standard MDP tasks (Barto, Sutton, and Anderson, 1983b; Russell and Norvig, 1994) can be specified by LTL formulas.

LTL has been used well in the domains of robotics and control systems (Wongpiromsarn, Topcu, and Murray, 2012; Kress-Gazit, Fainekos, and Pappas, 2009; Wolff, Topcu, and Murray, 2012; Lahijanian, Andersson, and Belta, 2011).

On the other hand, there are limitations when applying LTL in RL problems. The Simulation Lemma (Kearns and Singh, 1998), for example, says that, for any MDP and any  $\epsilon > 0$ , there exists an  $\epsilon' > 0$  such that finding optimal policies in an  $\epsilon'$ -close model of the real environment results in behavior that is  $\epsilon$ -close to optimal in the real environment. This property does not hold in the context of LTL tasks. In particular, there is an MDP and an  $\epsilon > 0$  such that no  $\epsilon'$ -close approximation for  $\epsilon' > 0$  is sufficient to produce a policy with  $\epsilon$ -close satisfaction probability (Littman et al., 2017).

## 2.4 "Short-Circuit" Geometric Linear Temporal Logic

To solve the problems of combining LTL and reward functions in RL settings, I present "Short-Circuit" Geometric Linear Temporal Logic, a special version of LTL. The brief idea of "Short-Circuit" Geometric Linear Temporal Logic (GLTL) is to add discounting factors to restrict the period of validity of the temporal operators. Similar ideas have been proved to be effective in prior work (Almagor, Boker, and Kupferman, 2014; De Alfaro, Henzinger, and Majumdar, 2003).

GLTL temporal operators expire after a geometrically distributed amount of time. An analogous version of the LTL formula  $\Box A$  is specified as  $\Box_{\mu} A$  in GLTL. It means that "A has to stay true for  $k$  time steps where  $k$  is a random variable following a geometric distribution with parameter  $\mu$ ."

Similarly, the LTL formula  $\Diamond A$  becomes  $\Diamond_{\mu} A$ , which means "A becomes true in the next  $k$  steps where  $k$  is a random variable following a geometric distribution with parameter  $\mu$ ." An LTL formula like  $A \mathcal{U} B$  is expressed in GLTL as  $A \mathcal{U}_{\mu} B$ , where  $A$  has to stay true at least until  $B$  becomes true, and  $B$  needs to become true in the next  $k$  steps, where  $k$  is a random variable following a geometric distribution with parameter  $\mu$ .

The geometric decay of GLTL settings can be considered as a generalization of discounted reward as commonly used in MDP-based decision making. The advantage of GLTL over LTL is very straightforward: It allows infinite tasks to be executable within a finite observation bounding window.

This difference is significant in a real-world problem where the speed at which a task is accomplished is important.

For example, when a housekeeper gives the command 'go back to the charger' to the vacuum, the task might be interpreted as  $\Diamond$ charger. In an LTL setting, such formula does not necessarily require the agent to find the shortest path to the charger since it's true as long as any time in the infinite future the state "agent is currently at the charger" becomes true. Therefore, the agent can perform whatever actions it chooses and will not be considered as violating the goal of the task. From a human perspective, it is weird to see the vacuum running everywhere randomly, but its behavior cannot be considered as wrong in LTL since it is possible that it may come to its charger sometime in the future.

In the GLTL setting, however, 'go back to the charger' is translated as  $\Diamond_{\mu}$ charger. That means the state "agent is currently at the charger" has to become true in the next  $k$  steps, where  $k \sim G1(\mu)$ . The vacuum will then find the optimal path to the charger to satisfy the GLTL specification.

'Stay at the charger, don't walk around' is another reasonable command a person might give. In the LTL setting, it could be expressed as  $\Box$ charger, which will never be true in the real world due to its requirement of needing to be true in the infinite future. This kind of tasks leads to other problems. For example, in the training process, it is impossible for the agent to learn such task a, since it can never be finished. When the trainer is a person, giving endless feedback to the robot to teach a simple task is not acceptable. In the GLTL setting, where the goal is specified as  $\Box_{\mu}$ charger, the same task can be trained within  $k$  steps,  $k \sim G1(\mu)$ .

A GLTL formula can be converted to an MDP-compatible formula recursively (Littman et al., 2017). We review the basic construction from GLTL formulas to *specification MDPs* here. Given a group of GLTL formulas  $\phi$ ,  $\phi_1$ ,  $\phi_2$ , and set of propositions  $P$  that can be evaluated on  $S$ , we have:

- For  $p \in P$ ,  $M_p = \langle s, acc, rej, A, T, R \rangle$  can be constructed that
  - if  $p$  is true at  $s$ ,  $T(s, a, acc) = 1$ ;
  - if  $p$  is false at  $s$ ,  $T(s, a, rej) = 1$ ;
- For  $\neg\phi$ ,  $M_{\neg\phi} = M_{\phi}$  with  $acc$  and  $rej$  swapped;



- For  $\phi_1 \wedge \phi_2$ ,  $M_\phi$  can be constructed from  $M_{\phi_1} = \langle S_1, A_1, T_1, R_1 \rangle$  and  $M_{\phi_2} = \langle S_2, A_2, T_2, R_2 \rangle$  where  
 $S = (S_1 \setminus \{r_1\}) \times (S_2 \setminus \{r_2\}) \cup \{r\}$ ,  $acc = (acc_1, acc_2)$ ,  
 $A = A_1 \times A_2$ ,  
if  $s'_1 = rej_1$  for  $(s_1, a_1, s'_1) \in M_{\phi_1}$  or  $s'_2 = rej_2$  for  $(s_2, a_2, s'_2) \in M_{\phi_2}$ ,  
 $T((s_1, s_2), (a_1, a_2), rej) = T_1(s_1, a_1, s'_1)T_2(s_2, a_2, s'_2)$ ,  
otherwise  $T((s_1, s_2), (a_1, a_2), (s'_1, s'_2)) = T_1(s_1, a_1, s'_1)T_2(s_2, a_2, s'_2)$ ;
- $\phi_1 \vee \phi_2 = \neg((\neg\phi_1) \wedge (\neg\phi_2))$ ;
- For  $\diamond_\mu\phi_1$ ,  $M_{\diamond_\mu\phi_1}$  can be constructed from  $M_{\phi_1} = \langle S_1, A_1, T_1, R_1 \rangle$  where  
 $S = S_1$ ,  $s = s_1$ ,  $acc = acc_1$ ,  $rej = rej_1$ ,  
 $A = A_1$ ,  
and the transitions of  $M_{\diamond_\mu\phi_1}$  can be constructed from the transitions of  $M_{\phi_1}$  shown in Table 2.1;
- $\square_\mu\phi_1 = \neg(\diamond_\mu(\neg\phi_1))$ ;

$s'_1$	$s'$	$p(s' s'_1)$
$acc_1$	$acc_1$	1
$rej_1$	$s_1$	$\mu$
	$rej_1$	$1 - \mu$
$s_1 \setminus \{acc_1, rej_1\}$	$s'_1$	$\mu$
	$rej_1$	$1 - \mu$

TABLE 2.1: Transition  $(s, a, s')$  in  $MDP_{\diamond_\mu\phi_1}$  constructed from a transition

$$(s_1, a_1, s'_1) \text{ in } M_{\phi_1}. \text{ As above, } p(s'|s'_1) = \frac{T(s, a, s')}{T_1(s_1, a_1, s'_1)}.$$

More details such as  $MDP_{\phi_1 \cup \mu\phi_2}$  can be found in (Littman et al., 2017).

The GLTL definitions above, although is already a reasonably significant improvement from the classical LTL settings, still could not handle real-world problems in some cases. A useful example is

$$\square(a \wedge \diamond b)$$

Under the GLTL semantics, for a sequence to satisfy the formula, it would need to include states where  $a$  is consistently true and  $b$  is true at least sporadically.

Therefore I introduce the "Short - Circuit" GLTL semantics: This semantics matches the GLTL semantics for simple formula like  $\diamond a$ . However, they differ for nested temporal operators. For the formula

$$\square(a \wedge \diamond b)$$

Under the short-circuit semantics, the value of  $a$  is only checked in the beginning and then each time  $b$  is checked. Thus,  $a$  need not always be true, but can be sporadically true if it is synchronized with  $b$ .

The meaning of an expression  $\phi$  in the context of a sequence position  $t$  is a pair, consisting of its truth value (first component) and the sequence position once that truth value is determined (second component).

The first component of such a pair  $\mathbb{V}(\phi, t)$  is denoted with  $\mathbb{V}(\phi, t).1$  and the second component is denoted with  $\mathbb{V}(\phi, t).2$ . These values are random variables, so we take the expectation of the truth value to be the overall value of the expression:  $\mathbb{E}[\mathbb{V}(\phi, 0).1]$ .

Here is how  $\mathbb{V}$  is defined:

$$\mathbb{V}(\mathbf{tt}, t) = \langle 1, t \rangle \quad (2.1)$$

$$\mathbb{V}(a, t) = \begin{cases} \langle 1, t \rangle & \text{if } w^t(0) \text{ maps } a \text{ to } \mathbf{tt}, \\ \langle 0, t \rangle & \text{otherwise.} \end{cases} \quad (2.2)$$

$$\mathbb{V}(\neg\phi, t) = \langle 1 - v, s \rangle \quad (2.3)$$

$$\mathbb{V}(\phi_1 \wedge \phi_2, t) = \begin{cases} \langle 0, \min(s_1, s_2) \rangle & \text{if } v_1 = 0 \wedge v_2 = 0, \\ \langle 0, s_1 \rangle & \text{if } v_1 = 0 \wedge v_2 = 1, \\ \langle 0, s_2 \rangle & \text{if } v_1 = 1 \wedge v_2 = 0, \\ \langle 1, \max(s_1, s_2) \rangle & \text{if } v_1 = 1 \wedge v_2 = 1. \end{cases} \quad (2.4)$$

$$\mathbb{V}(\phi_1 \vee \phi_2, t) = \mathbb{V}(\neg(\neg\phi_1 \wedge \neg\phi_2), t) \quad (2.5)$$

$$\mathbb{V}(\diamond\gamma\phi, t) = \begin{cases} \langle 0, k \rangle & \text{if } k < s, \\ \langle 1, s \rangle & \text{if } k \geq s \wedge v = 1, \\ \mathbb{V}(\diamond\gamma\phi, s + 1) & \text{otherwise.} \end{cases} \quad (2.6)$$

$$\mathbb{V}(\square\gamma\phi, t) = \mathbb{V}(\neg(\diamond\neg\phi), t) \quad (2.7)$$

$$\mathbb{V}(\phi_1 \mathcal{U}_\gamma \phi_2, t) = \begin{cases} \langle 0, k \rangle & \text{if } k < s_1 \leq s_2, \\ \langle 1, k \rangle & \text{if } k = s_1 = s_2 \wedge v_2 = 1, \\ \langle 0, k \rangle & \text{if } k = s_1 = s_2 \wedge v_1 = 0 \wedge v_2 = 0, \\ \mathbb{V}(\phi_1 \mathcal{U}_\gamma \phi_2, k + 1) & \text{if } k = s_1 = s_2 \wedge v_1 = 1 \wedge v_2 = 0, \\ \mathbb{V}(\phi_1 \mathcal{U}_\gamma \phi_2, k + 1) & \text{if } s_1 \leq k < s_2 \wedge v_1 = 1, \\ \langle 0, s_1 \rangle & \text{if } s_1 \leq k < s_2 \wedge v_1 = 0, \\ \langle 1, s_2 \rangle & \text{if } s_1 < s_2 \leq k \wedge v_1 = 1 \wedge v_2 = 1, \\ \mathbb{V}(\phi_1 \mathcal{U}_\gamma \phi_2, k + 1) & \text{if } s_1 < s_2 \leq k \wedge v_1 = 1 \wedge v_2 = 0, \\ \langle 0, s_1 \rangle & \text{if } s_1 < s_2 \leq k \wedge v_1 = 0, \\ \langle 0, k \rangle & \text{if } k < s_2 \leq s_1, \\ \langle 1, s_2 \rangle & \text{if } s_2 \leq k \wedge s_2 < s_1 \wedge v_2 = 1, \\ \langle 0, s_2 \rangle & \text{otherwise.} \end{cases} \quad (2.8)$$

$$\mathbb{V}(\phi_1 \rightarrow \phi_2, t) = \mathbb{V}((\neg\phi_1) \vee \phi_2, t) \quad (2.9)$$

$$\mathbb{V}(\bigcirc\phi, t) = \mathbb{V}(\phi, t + 1) \quad (2.10)$$

Here,

$$v = \mathbb{V}(\phi, t).1,$$

$$s = \mathbb{V}(\phi, t).2,$$

$$v_1 = \mathbb{V}(\phi_1, t).1,$$

$$s_1 = \mathbb{V}(\phi_1, t).2,$$

$$v_2 = \mathbb{V}(\phi_2, t).1,$$

$$s_2 = \mathbb{V}(\phi_2, t).2,$$

$k$ ,  $k_1$ , and  $k_2$  are chosen geometrically with parameter  $\gamma$ .

## 2.5 Samples

Here are a few samples to demonstrate how the "Short - Circuit" GLTL semantics apply to real-world problems:

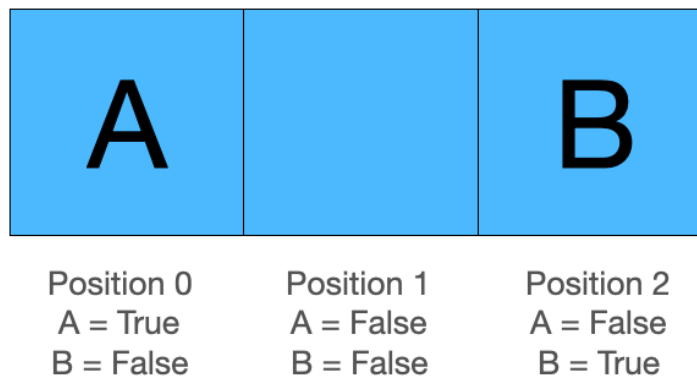


FIGURE 2.1: The simple 1 by 3 grid world.

Figure 2.1 illustrates a 1 by 3 simple grid world environment that contains two elements 'A' and 'B'. Here the two elements are the atomic propositions of this environment.

- At the very left cell, which is position 0, A is true and B is false.
- At the center cell, which is position 1, both A and B are false.
- At the very right cell, which is position 2, B is true and A is false.

Consider the following formula:

$$\diamond(A \wedge \diamond B)$$

It requires the agent to reach to the cell where A is true, then go to the cell where B is true, wherever it starts.

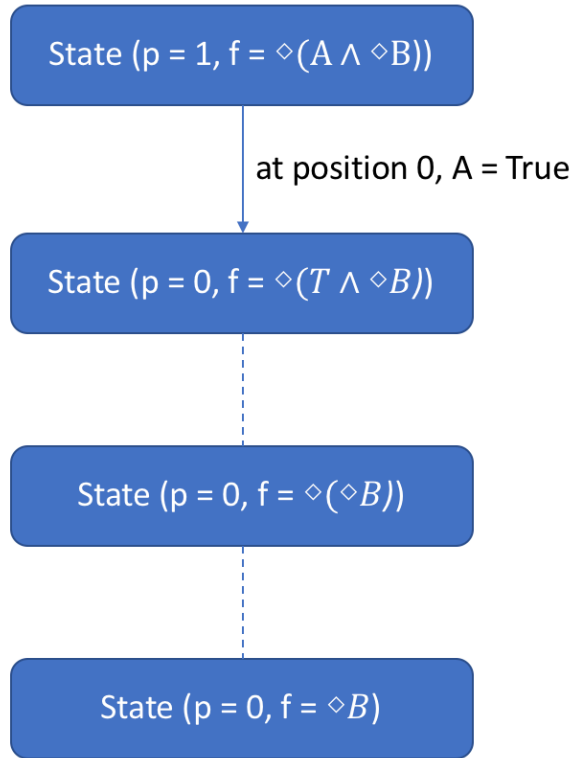


FIGURE 2.2: The change of status of the formula eventually (A and eventually B) at position 0 where A = True, B = False.

Figure 2.2 shows the change of the "status" of the formula assuming the agent is at position 1, the center cell initially:

- At the very first state,  $s_0$ , where the agent is at position 1, the state information can be represented as

$$p = 1, formula = \diamond(A \wedge \diamond B)$$

where  $p$  stands for position;

- The agent follows the policy of the MDP which is converted from the formula and goes left to position 0. Now the state information becomes

$$p = 0, formula = \diamond(True \wedge \diamond B)$$

- According to the rule of GLTL,

$$p = 0, formula = \diamond(True \wedge \diamond B)$$

is equivalent to

$$p = 0, formula = \diamond(\diamond B)$$

- 

$$p = 0, formula = \diamond(\diamond B)$$

is equivalent to

$$p = 0, formula = \diamond B$$

similarly

- The agent will then move right to position 1 at the next step, and then go on to position 2 following the expected behavior of the remaining status of the formula:  $\diamond B$ .
- When the agent arrives at position 2 finally, the whole task

$$\diamond(A \wedge \diamond B)$$

is finished.

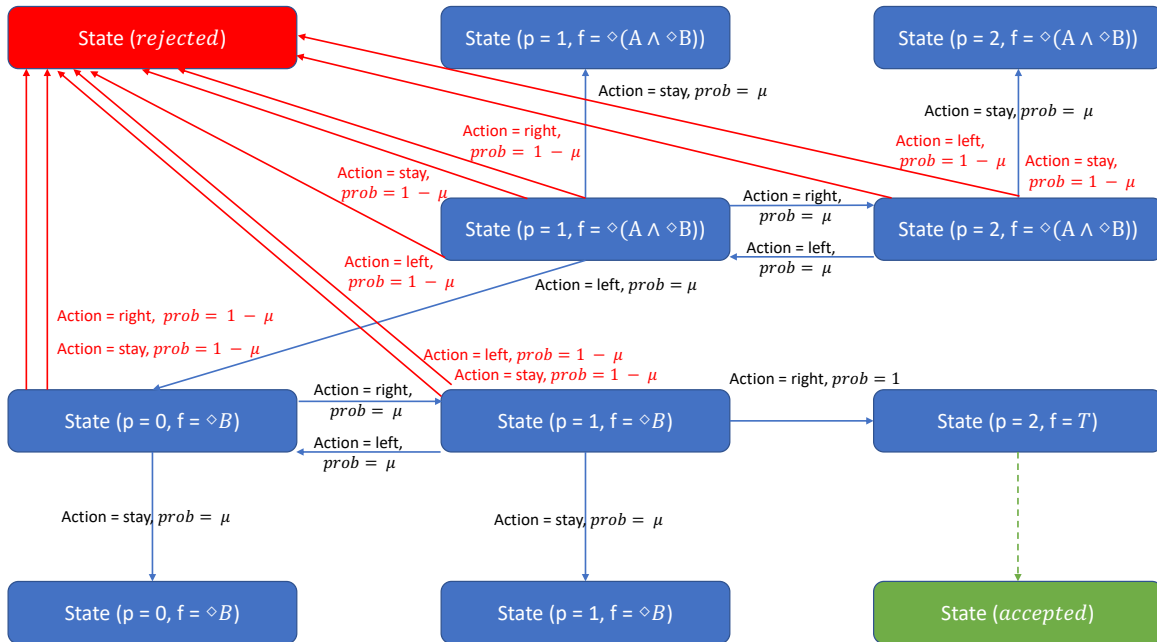


FIGURE 2.3: The transitions between states of the formula eventually (A and eventually B) in the simple 1 by 3 grid world.

The detailed overall transition map of this environment is presented by Figure 2.3. In any given environment, there is an extended space which creates a virtual state of "accepted" (marked in green at the bottom right of the graph) that represents the  $S_a$  in the GLTL definition. In the real-world problem, the state of "accepted" means that the entire GLTL formula is considered to be finished correctly.

There is also a virtual state of "rejected" (marked in red at the top left of the graph) that represents the  $S_r$  in the definition, which means the GLTL formula can not be satisfied from the current state. For every possible transition between two states, we use a function of  $\mu$  to denote the probability of the state becoming terminated as "accepted" or "rejected". In Figure 2.3, the red arrows and captions represent the transitions to the state of "rejected", and the green arrow represents the transition to the "accepted" state.

## 2.6 Conclusion

In contrast to standard MDP reward functions, we have provided a "Short-Circuit" Geometric Linear Temporal Logic (GLTL) semantics, an environment-independent specification for tasks. We have shown that this specification language can capture standard tasks used in the MDP

community and that it can be automatically incorporated into an environment MDP to create a fixed MDP to solve. Maximizing reward in this resulting MDP maximizes the probability of satisfying the task specification.

## Chapter 3

# Interactive Learning through Evaluative Human Feedback

This chapter investigates the problem of interactively machine learning from evaluative human feedback. The fundamental idea comes from the way that dogs get trained – human trainers use treats and alert tools during training, give the dog treat for performing well, and alert the dog when the it does something wrong. Most existing work in this domain are based on the assumption that the feedback that human trainers give depends on the desired behavior of the goal of the task, and has nothing to do with the current policy of the learner. In this chapter, I ’ll show empirical results that lead to the opposite opinion, that human feedback is policy-dependent.

Based on the insight discovery, I present *Convergent Actor-Critic by Humans* (COACH), an algorithm that could learn such policy-dependent feedback. COACH assumes that the *advantage function* (a value roughly corresponding to how much better or worse an action is compared to the current policy) provides a better model of human feedback, capturing human-feedback properties like diminishing returns, rewarding improvement, and giving 0-valued feedback a semantic meaning that combats forgetting (MacGlashan et al., 2017).

A series of comparative experiments in simulated environments were conducted to further show that COACH could perform well on goal feedback and stationary feedback, while classical algorithms, i.e., Q-Learning, TAMER can only learn from one or two types of feedback. To test whether COACH can learn from a real human trainer who gives evaluative numerical feedback, an experiment with a physical robot was done at last.

### 3.1 Human-Centered Reinforcement Learning

Traditionally, feedback in reinforcement learning usually comes from a hand-designed or automatically-constructed (Singh, Lewis, and Barto, 2009) reward function. Reward functions are the most common representations for tasks, mapping state features to scalar values (Sutton and Barto, 1998b; Littman, 2015). They can be learned from expert demonstration via inverse reinforcement learning (IRL) (Ng and Russell, 2000; Abbeel and Ng, 2004; Ziebart et al., 2008; Babes et al., 2011).

*Human-Centered Reinforcement-Learning* (HCRL) aims at solving problems of which the environment can be specified as an MDP while the rewards are not defined by the environment or the goal –it comes from a real human trainer.

Most existing RL approaches pale when the signals of reward come from humans and have largely failed to benefit from the sophisticated training strategies that expert animal trainers use with animals.

This failure has led to the development of new RL algorithms that are designed to learn from human-generated rewards and investigations into how people give interactive feedback (Knox and Stone, 2009a; Thomaz and Breazeal, 2006; Thomaz and Breazeal, 2007; Thomaz and Breazeal, 2008; Griffith et al., 2013; Loftin et al., 2015).

(Cruz et al., 2016) employed an IRL approach for the domestic task of cleaning a table. They compared three different methods and learned that the level of consistency of feedback was important. Training robotic agents with interactive feedback and contextual affordances presented an advantage over classic RL in terms of number of performed actions and collected reward. They claimed that the agent was able to learn the proposed cleaning task even when receiving wrong or inconsistent feedback in some time steps during the learning process.

(Krening and Feigh, 2019) proposed to investigate how the design of the interaction method for a Bayesian Q-Learning algorithm impacts human's experience of teaching the agent. They conducted a human-in-the-loop experiment in which people trained two agents with different teaching methods (critique and action advice) but the same underlying RL algorithm. Their results showed that an agent that learned from action advice created a better user experience compared to an agent that learns from critique. According to the results, they determined nine main characteristics of an algorithm's design that impact the human's experience with the agent, including using human instructions about the future, compliance with input, empowerment, transparency, immediacy, a deterministic interaction, the complexity of the instructions, accuracy of the speech recognition software, and the robust and flexible nature of the interaction algorithm.

The majority of these algorithms are designed based on the assumption that human feedback only depends on the desired behavior of the task regardless of the actual performance or policy of the learner.

In an abstractly formalized situation, trainers use positive numeric feedback as reward and negative numeric feedback as punishment to teach target policies. If feedback is stationary and intended to be maximized, it can be treated as a reward function that standard RL algorithms could use. Experiment in this chapter, on the other side, finds different results. A few approaches have also found similar phenomenon (Ho et al., 2015; Knox et al., 2012; Isbell et al., 2001). I'll demonstrate why most previous algorithms suffer in this situation through empirical results.

## 3.2 Human Feedback is Policy-Dependent

Previous approaches commonly assume that human trainer normally gives stationary feedback depending on only the quality of an agent's selection of its actions, thus the human feedback can be considered as a reward function. They proposed that standard reinforcement learning algorithms can be used in the way that the agent could find an optimal policy to maximize the rewards it could receive in the process (Pilarski et al., 2011; Isbell et al., 2001).





FIGURE 3.1: The goal of the task is to teach the robot to move to the red flag at the bottom left corner. The human trainer gives big positive reward when the robot moves closer to the flag, and small negative reward when it moves away, which leads to a positive reward cycle.

On the other hand, if general human feedback is policy-dependent, algorithms that rely on the wrong assumption may result in unexpected learning performance. In some cases where non-expert human trainers tend to be "over friendly", some of the feedback could form a *positive reward cycles* as shown in Figure 3.1.

A positive reward cycle is a special feedback pattern that is maximized by a looping behavior that endlessly accrues a net positive return. When a positive reward circle occur in the training process, the agent will likely converge to local optimal result instead of global optimal policy. It can cause the agent to perform unintended behaviors due to unanticipated exploits (Knox, 2012; Ho et al., 2015).

One possible reason to the problem is that people tend to give more reward to the agent for improving its performance than maintaining the previous status. Moreover, if the agent maintains the same level of performance for a long time without showing evidence of improvement, the human trainer may become disappointed and give 0 reward or even negative reward to the same behavior over time. It's common in the dog training scenario where a young puppy can easily get huge treat for learning how to 'sit', but a well trained adult dog may get nothing for doing the same thing.

Another example can be observed in k-12 education, that if a student whose past GPA was C may feel very happy to get a B, but a student with A+ GPA may feel bad with a B. The same behavior, getting a B, could receive different feedback in the same environment. Such type of feedback is considered as policy dependent. A policy-dependent feedback is decided by the current policy of the learner, that how better or worse the learner's current action is compared to his previous performance.

In general, human feedback can be considered to have three properties (MacGlashan et al., 2017):

- Diminishing returns.

As the learner performs better over time, the positive feedback from human trainers

decrease because they think "it's already been learned".

- Differential feedback.  
The strength of feedback varies with improvement and deterioration.
- Policy shaping.  
It reinforces sub-optimal actions, then punishes them and raises the bar.

### 3.2.1 Empirical Results

Click 'Go' to start today's training.

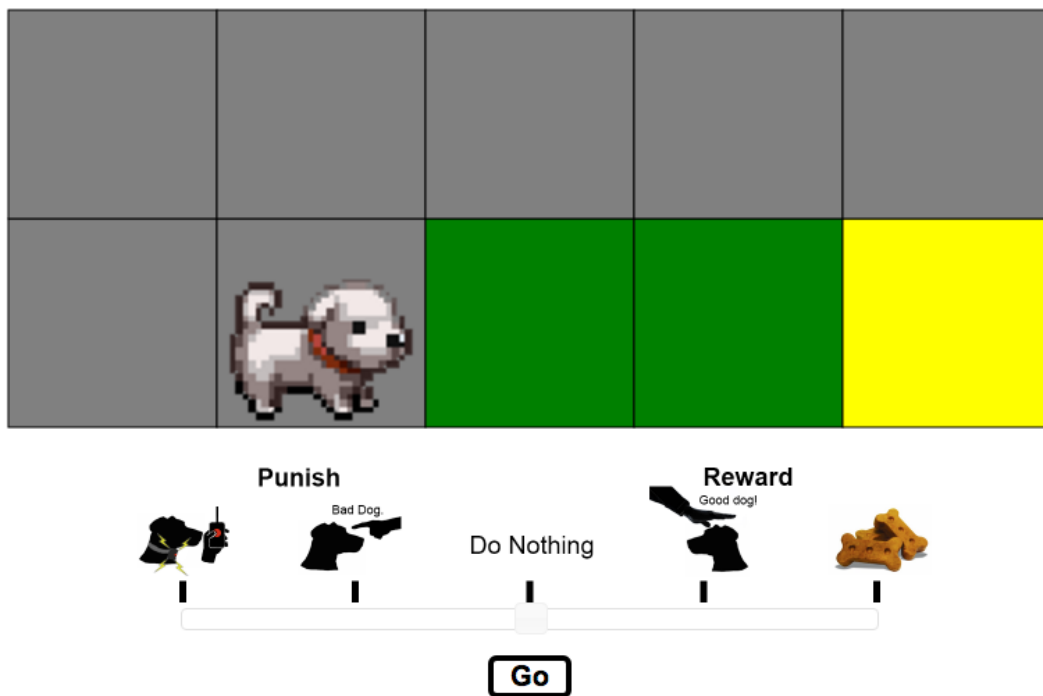


FIGURE 3.2: The training interface that was shown to the participants during the online study.

We conducted an online experiment on Amazon Mechanical Turk (AMT). Participants were recruited through AMT, and signed the research consent before the official experiment (MacGlashan et al., 2017).

Participants were shown an user interface as displayed in Figure 3.2, which was a 2 by 5 grid world. For each *episode*, a dog always started from the cell on the right of the bottom left corner, and went to the destination. There were two green cells representing grass, which the dog should avoid touching. The destination is at the bottom right corner colored in yellow.

Participants were asked to train the dog to find a way to go to the yellow destination as fast as possible. They were also told that the dog should never touch the green cells. In the orientation, they were informed that as a result of prior training, their dog had already learned one of the three behaviors, which were “bad”, “alright”, and “good”.

In all cases, the dog would start from the same location shown in Figure 3.2:

- “Bad” dogs went right and walked straight through the green cells to the yellow cell.
- “Alright” dogs first moved left, then up, and then to the goal, avoiding green but not taking the shortest route.

- “Good” dogs took the shortest path to yellow without touching the green cells.

All participants were randomly assigned to three groups. Before the training got started, participants watched the behaviors that their dogs had learned, which was one of the three conditions depending on which group the participant belonged to.

During training, participants saw the dog take an action, and then gave feedback after every action using a continuous labeled slider as shown. The slider always started in the middle of the scale on each trial, and several points were labeled with different levels of reward (praise and treats) and punishment (scolding and a mild electric shock). Participants went through a brief tutorial using this interface. Responses were coded as a numeric value from  $-50$  to  $50$ , with “Do Nothing” as the zero-point (MacGlashan et al., 2017).

During the training phase, participants trained a dog for three *episodes*. The first two episodes were same as the learned behavior that had been told to the participants. The last episode, regardless of which group the participants belonged to, were all an ‘Alright’ track.



FIGURE 3.3: The three conditions of the dog in three consecutive episodes.

Therefore, each user could see one of three different conditions as shown in Figure 3.3:

- “Improving”:  
The dog performed “Bad” behavior in the first two episodes, and then behaved “Alright” in the third episode, which formed an “improving” condition.
- “Steady”:  
The dog performed “Alright” behavior in all three episodes.
- “Degrading”:  
The dog performed “Good” behavior in the first two episodes, then behaved in an “Alright” way in the third episode.

Because the behavior is identical in the final episode for all conditions, if human feedback is independent from the learner’s policy, the expectation of feedback of all three groups would be the same since the behavior that the participants gave feedback to were exactly the same.

However, if feedback is policy dependent, we would expect more positive feedback for the improving condition.

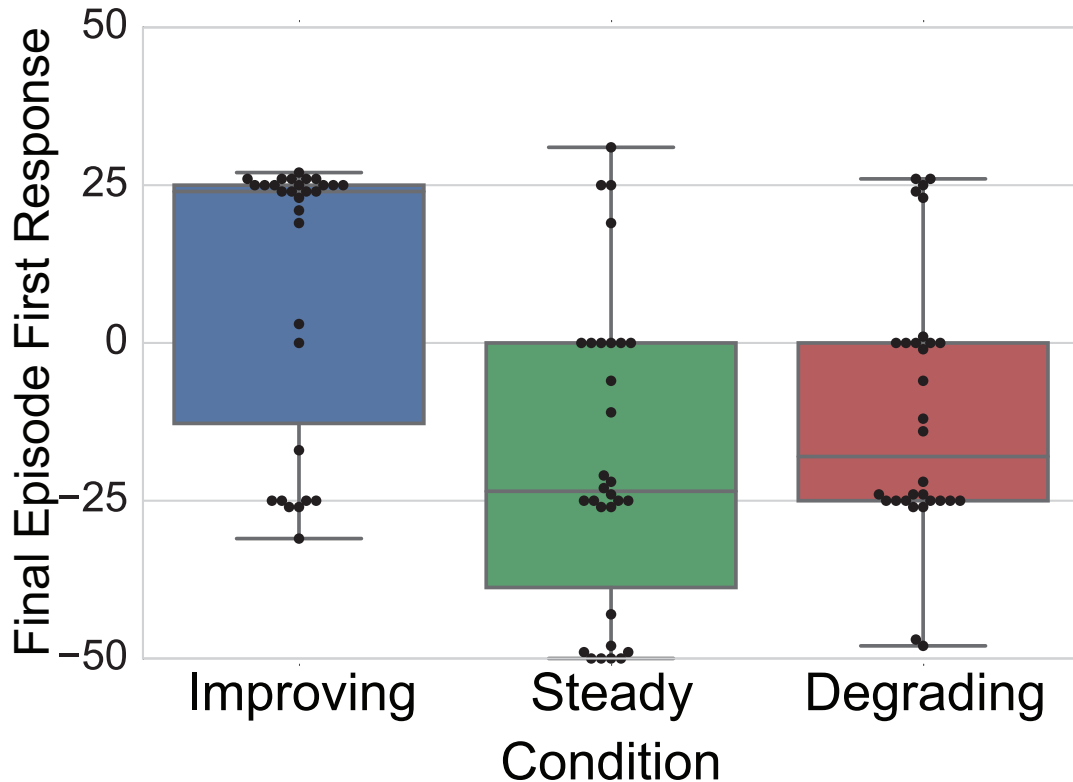


FIGURE 3.4: The feedback distribution for first step of the final episode for each condition. Feedback tended to be positive for improving behavior, but negative otherwise.

The boxplots results are displayed in Figure 3.4. The "Improving condition" group are shown in the blue area, the "Steady condition" group are in green area, and the "Degrading condition" group are in red area. The vertical axis represents the first response that the participants gave to the last episode they watched.

We can see that even though the behaviors of the dogs in the last episode were exactly the same to all participants of all three groups, the "Improving condition" received significantly more positive feedback than the other two conditions. The results clearly supported our hypothesis that human feedback is policy-dependent.

### 3.3 Convergent Actor-Critic by Humans

Based on the discovery that human feedback is policy dependent, we used the *Advantage Function* (Baird, 1995) to simulate human feedback.

The advantage function  $A^\pi$  is defined as

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \quad (3.1)$$

The advantage function measures how much better or worse an action selection is compared to the agent's performance under its current policy  $\pi$ .

According to the policy gradient algorithm introduced by (Sutton et al., 1999), we have:

$$\Delta\theta = \alpha \nabla_{\theta} \rho = \alpha \sum_s d^\pi(s) \sum_a \nabla_{\theta} \pi(s, a) Q^\pi(s, a),$$

where

- $s$  represents the state information;
- $a$  represents the action selection at  $s$ ;
- $\theta$  is the parameters that decide the agent's behavior;
- $\alpha$  is the step size, also known as the learning rate;
- $\rho$  is the discounted expected reward from a fixed start state distribution;
- $d^\pi(s)$  is the component of the stationary distribution at  $s$ ;

Therefore, the update of the policy  $\pi$  at time  $t$  can be denoted as:

$$\Delta\theta_t = \alpha_t \nabla_{\theta} \pi(s_t, a_t) \frac{f_{t+1}}{\pi(s_t, a_t)}, \quad (3.2)$$

Here,  $f_{t+1}$  is the feedback from the trainer, which is real people in this problem. If the human feedback is policy dependent, we have

$$f_t = Q^\pi(s_t, a_t),$$

and

$$E[f_{t+1}] = Q^\pi(s_t, a_t) - v(s),$$

we get a convergent learning algorithm.

Consider feedback

$$f_t = Q^\pi(s_t, a_t) - V^\pi(s_t) = A^\pi(s_t, a_t),$$

It will also converge since the trainer needs to "guess" the current policy  $\pi$  of the learner through evaluating the actions that the agent take from each state.

Comparing real human feedback to the advantage function, we can see that the advantage function is consistent with the three properties of human feedback;

- Diminishing returns.  
As the policy  $\pi$  of the learner gradually optimizes over time, the potential space for further improvement decreases, therefore the simulated feedback from the advantage function will diminish. If the policy becomes optimal, which means there is no room for improvement any more, then the simulated feedback generated by the advantage function becomes zero or negative.
- Differential feedback.  
The advantage function demonstrates the magnitude of improvement of an action over the current policy of the learner, therefore the feedback will be differential similar to human feedback.
- Policy shaping.  
If the current action shows improvement over the current policy of the learner, then the simulated feedback is positive, otherwise it'll be negative.

Therefore it's reasonable to use the advantage function to simulate the policy-dependent human feedback. In reality, human feedback is richer and sometimes unpredictable, which can not be perfectly represented by the advantage function. I'm sure there can be other approaches

to simulate human feedback better, but it is not the main purpose of this chapter. The research of stronger simulation of real human feedback may be a promising topic for further research.

Moreover, real human feedback, if allowed by the learning algorithm, is normally not numerical. Most common people use much more natural language than positive or negative evaluative feedback to train their pets, this part is discussed in Chapter 5.

### 3.3.1 Real-time COACH

Applying Equation 3.2 to stationary environment is demonstrated in the comparison experiment, which will be presented in the next section. In real-world experiment with a physical robot, however, the robot moves in a continuous setting, which requires the problem of credit assignment to be well resolved (Knox and Stone, 2009a).

Due to reaction time, human feedback is typically delayed by about 0.2 to 0.8 seconds from the moment an action happens to the moment the agent (robot) receives feedback to that action (Knox, 2012). First of all, it takes time for people to observe an "action" due to the time cost of the robot finishing the desired behavior; Secondly, it's hard to tell when an action ends if the robot keeps moving without a stop break; Thirdly, sometime it takes a short while for the human trainer to decide what feedback should be given to the robot's behavior; Lastly, signal transmission takes time, too.

To handle this delay, feedback in Real-time COACH is associated with events from  $d$  steps ago to cover the gap.

Another important challenges in a continuous real-world problem is sparse feedback. It's quite often when the human trainer could not tell the beginning and end of an action. When the robot keeps moving, a human trainer may give feedback when he thinks the robot has finished doing something and is waiting for his command. On the other hand, according to the diminishing return property of human feedback, many people tend to give feedback less frequently over time.

In order to handle the challenges, we introduce *Real-time COACH 1*:

---

#### Algorithm 1 Real-time COACH

---

**Require:** policy  $\pi_{\theta_0}$ , trace set  $\lambda$ , delay  $d$ , learning rate  $\alpha$

Initialize traces  $e_\lambda \leftarrow \mathbf{0} \forall \lambda \in \lambda$

observe initial state  $s_0$

**for**  $t = 0$  to  $\infty$  **do**

    select and execute action  $a_t \sim \pi_{\theta_t}(s_t, \cdot)$

    observe next state  $s_{t+1}$ , sum feedback  $f_{t+1}$ , and  $\lambda$

**for**  $\lambda' \in \lambda$  **do**

$$e_{\lambda'} \leftarrow \lambda' e_{\lambda'} + \frac{1}{\pi_{\theta_t}(s_{t-d}, a_{t-d})} \nabla_{\theta_t} \pi_{\theta_t}(s_{t-d}, a_{t-d})$$

**end for**

$$\theta_{t+1} \leftarrow \theta_t + \alpha f_{t+1} e_\lambda$$

**end for**

---

We choose reward aggregation for variable magnitude reward (Knox and Stone, 2009a), where a trainer selects from a discrete set of feedback values and further raises or lowers the numeric value by giving multiple feedbacks in succession that are summed together (MacGlashan et al., 2017).

*eligibility traces* (Barto, Sutton, and Anderson, 1983a) are used to assign credits to their target states and actions. An eligibility trace is a vector that keeps track of the policy gradient and decays exponentially with a parameter  $\lambda$ . Policy parameters are then updated in the direction of the trace, allowing feedback to affect earlier decisions.

Sparse feedback for real-time models means that the feedback to some states are zero, or neutral. According to the diminishing return property and differential feedback property of human feedback, neutral feedback does not mean there's no useful information for the algorithm to update its policy. When a human trainer gives no feedback, it could imply that he's okay with the current action selection of the agent comparing to the policy of the learner, meanwhile he has not seen improvement from previous performance.

For Real-time COACH, the Advantage Function suggests that neutral feedback may be a good signal that the learner is not doing worse than before.

In the pilot study, we observed that some trainers were very passionate to give feedback, that their feedbacks were too frequent for the algorithm, it can destabilize learning. Actually if feedback is equal to the sign of the advantage function:

- +1, if  $A^\pi(s, a) > 0$ ;
- -1, if  $A^\pi(s, a) < 0$ ;

then using eligibility traces can cause this destabilization.

For this reason, it can be useful to give the trainer two or more feedback channels that are associated with different eligibility trace  $\lambda$  values; one that is near zero for more frequent local feedback, and one with a larger  $\lambda$  for critiquing longer histories. This discovery contributes to later research presented in Chapter 5.

### 3.4 Comparison of COACH, Q Learning, and TAMER in Simulated Grid World

Start	-1	-1	-1	-1	-1	-1	Goal +5

FIGURE 3.5: The simulated  $8 \times 5$  grid in which the agent starts in 0,0 and must get to 7,0, which yields +5 reward. However, from 1,0 to 6,0 are cells the agent needs to avoid, which yield -1 reward.

To understand the behavior of COACH with different types of trainer feedback strategies, we carried out a controlled comparison in a simple grid world as is shown in Figure 3.5. The domain is essentially an expanded version of the dog domain used in our human-subject experiment. It is a  $8 \times 5$  grid in which the agent starts in  $0,0$  and must get to  $7,0$ , which yields  $+5$  reward. However, from  $1,0$  to  $6,0$  are cells the agent needs to avoid, which yield  $-1$  reward.

### 3.4.1 Learning Algorithms and Feedback Strategies

Three types of learning algorithms were tested. Each maintains an internal data structure: It updates with feedback of the form  $\langle s, a, f, s' \rangle$ , where  $s$  is a state,  $a$  is an action taken in that state,  $f$  is the feedback received from the trainer, and  $s'$  is the resulting next state. The algorithm also must produce an action for each state encountered.

The first algorithm, Q learning (Watkins and Dayan, 1992), represents a standard value-function-based RL algorithm designed for reward maximization under delayed feedback. It maintains a data structure  $Q(s, a)$ , initially 0. Its update rule has the form:

$$\Delta Q(s, a) = \alpha [f + \gamma \max_{a'} Q(s', a') - Q(s, a)]. \quad (3.3)$$

Actions are chosen using the rule:  $\arg \max_a Q(s, a)$ , where ties are broken randomly. We tested a handful of parameters and used the best values: discount factor  $\gamma = 0.99$  and learning rate  $\alpha = 0.2$ .

In TAMER (Knox and Stone, 2009b), a trainer provides interactive numeric feedback that is interpreted as an exemplar of the reward function for the demonstrated state–action pair as the learner takes actions. We assumed that each feedback applies to the last action, and thus used a simplified version of the algorithm that did not attempt to spread updates over multiple transitions. TAMER maintains a data structure  $R_H(s, a)$  for the predicted reward in each state, initially 0. It is updated by:  $\Delta R_H(s, a) = \alpha f$ . We used  $\alpha = 0.2$ . Actions are chosen via an  $\epsilon$ -greedy rule on  $R_H(s, a)$  with  $\epsilon = 0.2$ .

Lastly, we examined COACH, which is also designed to work well with human-generated feedback. We used a softmax policy with a single  $\lambda = 0$  trace. The parameters were a matrix of values  $\theta(s, a)$ , initially zero. The stochastic policy defined by these parameters was

$$\pi(s, a) = e^{\beta \theta(s, a)} / \sum_a e^{\beta \theta(s, a)},$$

with  $\beta = 1$ . Parameters were updated via

$$\Delta \theta = \alpha \nabla_{\theta} \pi(s, a) \frac{f}{\pi(s, a)}, \quad (3.4)$$

where  $\alpha$  is a learning rate. We used  $\alpha = 0.05$ .

In effect, each of these learning rules makes an assumption about the kind of feedback it expects trainers to use. We wanted to see how they would behave with feedback strategies that matched these assumptions and those that did not.

The first feedback strategy we studied is the classical task-based reward function (“task”) where the feedback is sparse:  $+5$  reward when the agent reaches the goal state,  $-1$  for avoidance cells, and 0 for all other transitions. Q-learning is known to converge to optimal behavior with this type of feedback.

The second strategy provides policy-independent feedback for each state–action pair (“action”):  $+5$  when the agent reaches termination,  $+1$  reward when the selected action



matches an optimal policy,  $-1$  for reaching an avoidance cell, and  $0$  otherwise. This type of feedback serves TAMER well.

The third strategy (“improvement”) used feedback defined by the advantage function of the learner’s current policy  $\pi$ ,  $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ , where the value functions are defined based on the task rewards.

Instead of providing this feedback signal on every state transition, we included a parameter  $\rho$  that determined the probability of delivering feedback. This parameter was meant to capture the idea that real human trainers are unlikely to provide feedback on every single step due to fatigue. We set  $\rho = 0.5$ . This type of feedback is very well suited to COACH.

### 3.4.2 Results

Each combination of algorithm and feedback strategy was run 99 times with the median value of the number of steps needed to reach the goal reported. Episodes were ended after 1,000 steps if the goal was not reached. The results are presented in Figure 3.6, Figure 3.7, Figure 3.8, and Figure 3.9, which show the steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis are on a logarithmic scale.

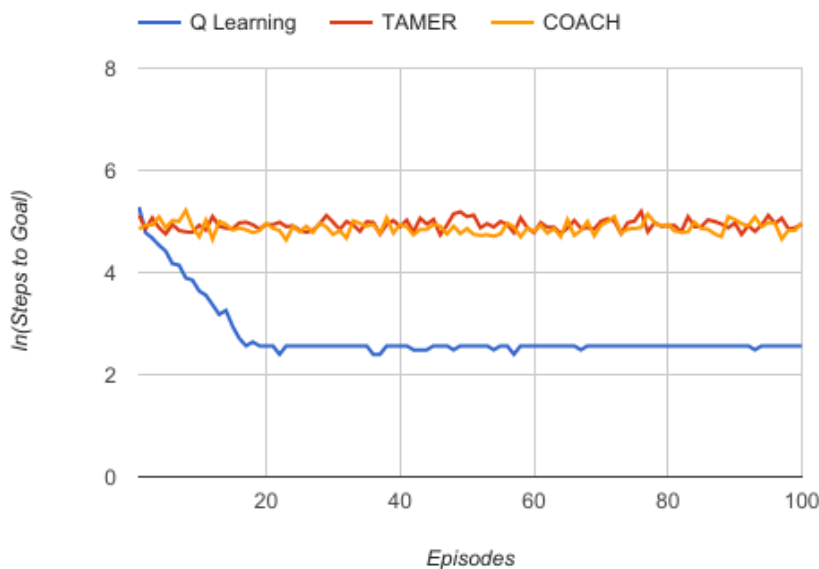


FIGURE 3.6: Task feedback. COACH is without eligibility traces. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale.

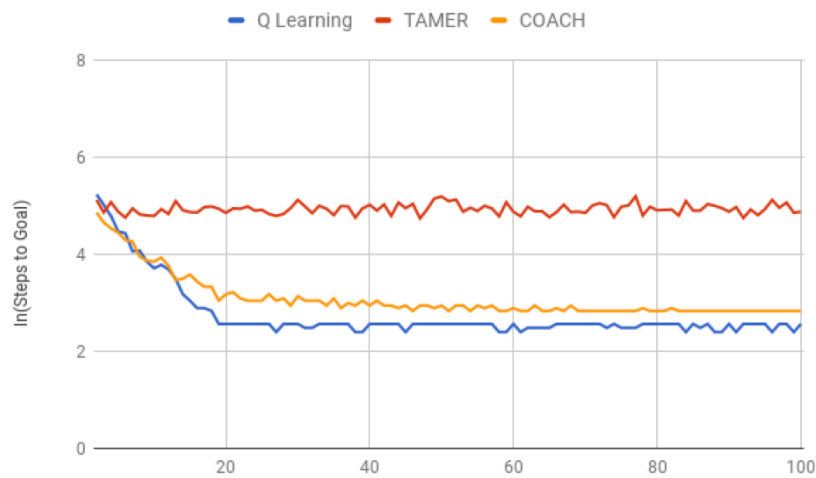


FIGURE 3.7: Task feedback. COACH is with eligibility traces. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale.

Figure 3.6 shows the steps needed to reach the goal for the three algorithms trained with task feedback. The figure shows that TAMER can fail to learn in this setting. COACH also performs poorly with  $\lambda = 0$ , which prevents feedback from influencing earlier decisions. We did a subsequent experiment (shown in Figure 3.7) with  $\lambda = 0.9$  and found that COACH converged to reasonable behavior, although not as quickly as Q learning. This result helps justify using traces to combat the challenges of delayed feedback.

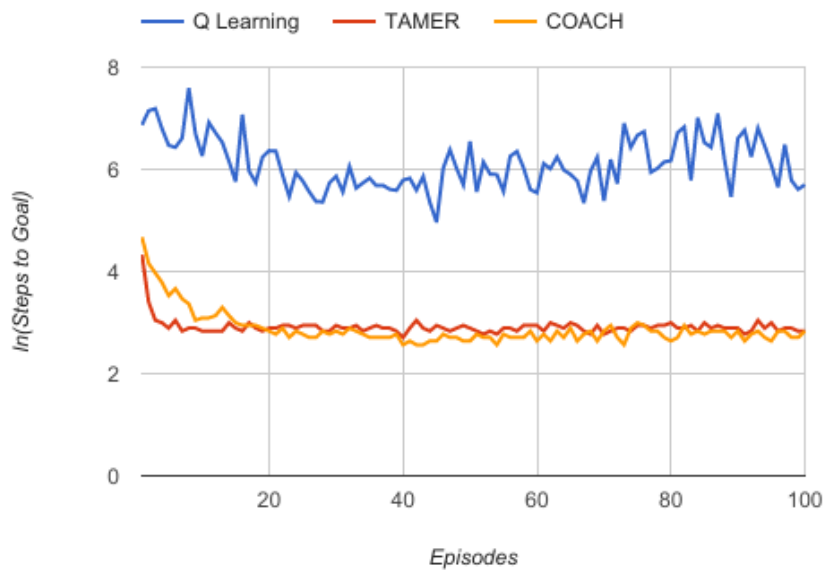


FIGURE 3.8: Action feedback. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale.

Figure 3.8 shows results with action feedback. This time, Q learning fails to perform well, a consequence of this feedback strategy inducing positive behavior cycles as it tries to avoid ending the trial, the same kind of problem that HCRL algorithms have been designed to avoid. Both TAMER and COACH perform well with this feedback strategy. TAMER performs slightly better than COACH, as this is precisely the kind of feedback TAMER was designed to handle.

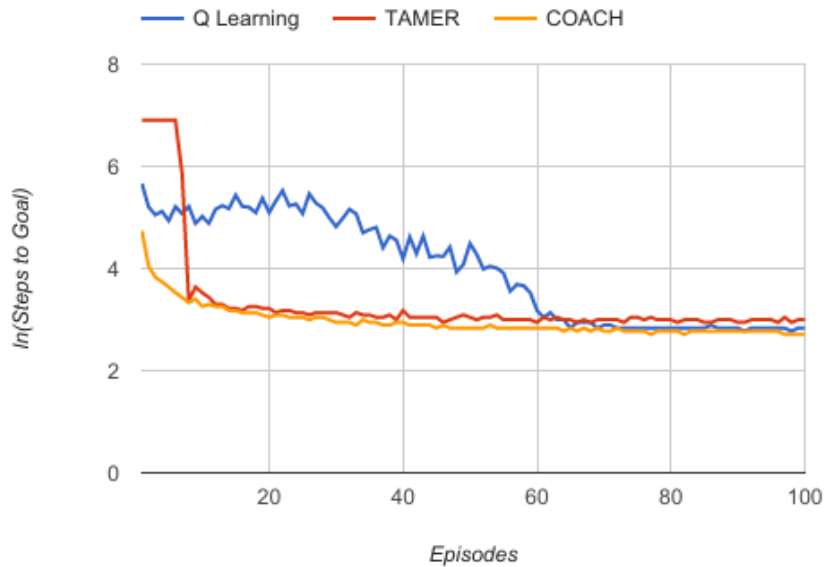


FIGURE 3.9: Improvement feedback. Steps to goal for Q learning (blue), TAMER (red), and COACH (yellow) in the grid world. The y-axis is on a logarithmic scale.

Figure 3.9 shows the results of the three algorithms with improvement feedback, which is generated via the advantage function defined on the learner's current policy. These results tell a different story. Here, COACH performs the best. Q-learning largely flounders for most of the time, but with enough training sometimes starts to converge. (Although, 14% of the time, Q learning fails to do well even after 100 training episodes). TAMER, on the other hand, performs very badly at first. While the median score in the plot shows TAMER suddenly performing more comparably to COACH after about 10 episodes, 29% of our training trials completely failed to improve and timed-out across all 100 episodes.

In the real world, human trainers are unlikely to always give feedback at each step. In many cases people give feedback at a much lower density. In an comparison between the performance of Q Learning, TAMER and COACH with "improvement" feedback strategy at different level of feedback density. We find out that TAMER is not robust when the feedback density changes, it does not converge when the feedback density is less than 60%, which is because TAMER relies on sufficient feedback. Q learning is robust, and it runs well when the feedback is sparse, while the performance decreases when more feedback is given by the trainer. COACH, on the other side, achieves a robust and good consistently good performance, it has maintained similar score (the number of steps to goal) since the feedback density reaches 30%.

### 3.5 Robot Case Study

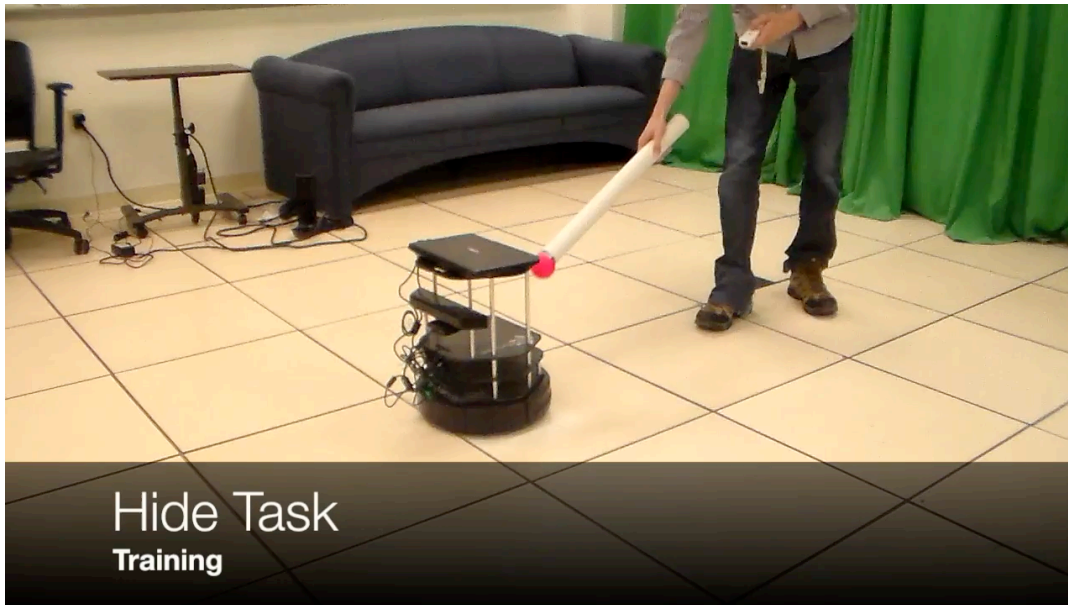


FIGURE 3.10: The TurtleBot experiment in the Rlab of Brown University. The human trainer is about to train the TurtleBot to learn to "hide" when it sees the pink ball.

We applied Real-time COACH on a physical TurtleBot robot (Amsters and Slaets, 2019) to further study whether COACH can scale to complex domain and enable real human trainer to teach a real robot tasks through evaluative feedback as shown in Figure 3.10.

The TurtleBot is a mobile base with two degrees of freedom that senses the world from a Kinect camera. We used the RGB image channels on the Kinect so that the target objects in the study: a pick ball, and a fixed cylinder with an orange top, can be recognized by the TurtleBot.

The action space of the TurtleBot is:

- move forward,
- move backward,
- rotate clockwise,
- rotate counterclockwise,
- stay (do nothing).

The agent decides an action to take every 33ms. A Nintendo Wii controller was used to give +1, +4, or -1 numeric feedback, pause, and continue training.

The features were constructed by first transforming the image into two color channels associated with the colors of the ball and cylinder. Sum pooling to form a lower-dimensional

$8 \times 8$  grid was applied to each color channel. Each sum-pooling unit was then passed through three different normalized threshold units defined by

$$T_i(x) = \min\left(\frac{x}{\phi_i}, 1\right),$$

where  $\phi_i$  specifies the saturation point. Using multiple saturation parameters differentiates the distance of objects, resulting in three “depth” scales per color channel. We selected the threshold values to detect our objects on 3 different relevant scales. Finally, we passed these results through a  $2 \times 8$  max-pooling layer with stride 1. (MacGlashan et al., 2017)

Five different tasks were designed and conducted:

- push - pull:

For the task “push - pull”, the TurtleBot was trained to navigate to the ball when it is far, and back away from it when it is near.

- hide:



FIGURE 3.11: In the training period of the “hide” task, the agent received a punishment signal, which was a negative feedback when it went near the pink ball. The red circle at the upper left shows the reward signal.

For the task “hide”, the goal is to teach the agent to move away from the ball. If the agent is distant from the ball, then the trainer needs to teach it to turn away from facing the ball.

The agent receives a negative feedback as punishment from the human trainer as is presented in Figure 3.11, where the red circle at the upper left of the screen shows the negative reward.



FIGURE 3.12: In the "hide" task training, the agent received a reward signal, which was a positive feedback when it went away from the pink ball. The blue square at the upper left shows the reward signal.

Figure 3.12 shows the agent receives a positive feedback when it goes away from the pink ball. The blue square at the upper left corner of the image shows the real-time reward signal given by the human trainer.



FIGURE 3.13: In the verification test of the "hide" task, the agent captured the pink ball and was about to behave according to the policy it had learned. No feedback was given by the human trainer during verification.

When the human trainer thought the agent had learned the task, he stopped training, and the agent was ready for the verification test. In the verification test, the agent received no feedback. Figure 3.13 and Figure 3.14 demonstrate the behavior of the agent, that it went backwards to get away from the pink ball.





FIGURE 3.14: In the verification test of the "hide" task, the agent saw the pink ball and went away from it, which was what it had been trained by the human trainer to do to "hide". No feedback would be given by the human trainer during verification.

- ball following:

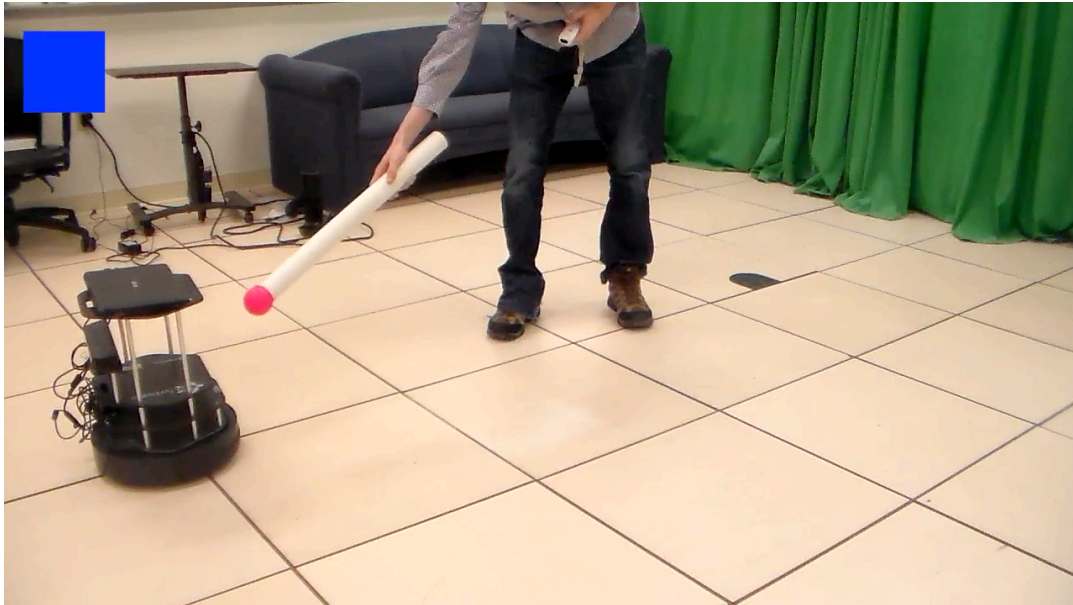


FIGURE 3.15: In the "ball following" task, the human trainer gave a positive feedback when the TurtleBot went to the pink ball.

For the "ball following" task, the goal is to teach the TurtleBot to move to the ball and follow it. Figure 3.15 shows that the human trainer gave a positive feedback to the agent when it went closer to the pink ball.

- alternate:



FIGURE 3.16: In the "alternate" task, the TurtleBot was trained to go back and forth between the pink ball and a cylinder.

For the "alternate" task, the agent was first trained to navigate to the ball when it saw the ball, and then turn away when it's near. Then the trainer replaced the ball with the cylinder and repeated the training process 3.16.

After the two rounds of training, the agent successfully learned to move to the ball when it appeared in its scene, then turned away when the ball's near. The agent learned the same behavior with the cylinder as well.

The trainer then placed both the ball and the cylinder in the environment, finding that the agent could move back and forth between the two objects.

- cylinder navigation:



FIGURE 3.17: In the "cylinder navigation" task, the human trainer used the pink ball as a lure to teach the agent to go to the cylinder.

For "cylinder navigation" task, we tried an animal-training method called *lure training*. In lure training, an animal is first taught to follow a lure object. Then the trainer uses the lure to guide the animal through the correct behavior and then give reward when the final goal is achieved. In this case, the goal is to teach the TurtleBot to learn to reach to the cylinder. The human trainer placed the cylinder and the pink ball together as shown in Figure 3.17, and tried to teach the agent to navigate to the cylinder with the help of the lure.

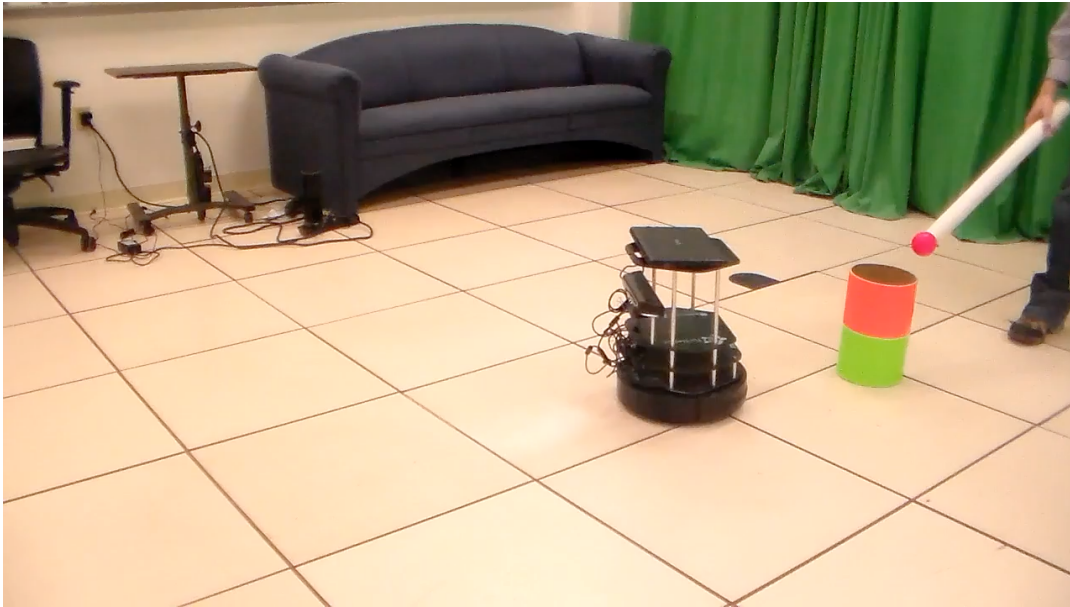


FIGURE 3.18: In the "cylinder navigation" task, the TurtleBot followed the pink ball as the lure and went towards the cylinder.

The pink ball was used as the lure in Figure 3.18. A +4 reward would be given to supportive actions along the way.



FIGURE 3.19: In the verification test of the "cylinder navigation" task, the TurtleBot could only see the cylinder. The pink ball lure did not appear in its scene.

Once the agent had learned to reach to the cylinder following the guidance of the lure, the pink ball, the trainer removed the pink ball from the scene of the TurtleBot's vision as shown in Figure 3.19, the TurtleBot could still find the way to the cylinder without the lure in Figure 3.20.

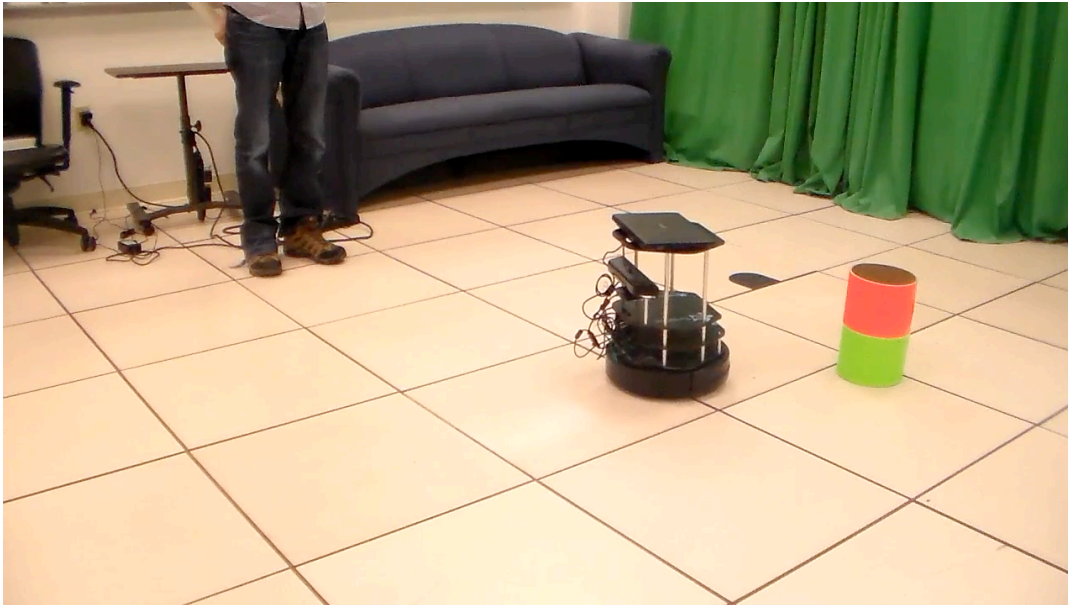


FIGURE 3.20: In the verification test of the "cylinder navigation" task, the TurtleBot navigated to the cylinder without seeing the pink ball lure.

We further classified training methods for each of these behaviors as:

- *flat*:  
the "push-pull", "hide", and "ball following" behaviors.
- *compositional*:  
the "alternate" and "cylinder navigation" behaviors.

We also applied TAMER to the TurtleBot for comparison purposes. To our knowledge when the experiment was conducted, it was the only HCRL algorithm without requiring feedback from non-human sources that could success on a similar platform (Knox, Stone, and Breazeal, 2013).

SABL (Loftin et al., 2014) and Policy Shaping were not used since those approaches have thus far only be demonstrated in tabular learning settings, whereas we make use of function approximation with the TurtleBot.

### 3.5.1 Results

A video about the experiment is available online at [https://youtu.be/\\_gt9npROvKQ](https://youtu.be/_gt9npROvKQ). COACH succeeded in learning all the five tasks. The total training and testing time cost for each task was less than two minutes. The COACH agent could learn complicated mission through learning its sub-parts separately and successfully learn the full mission through composition. Both the "alternate" mission and the "cylinder navigation" mission showed the advantages of the COACH model with eligibility traces.

TAMER, on the other hand, forgot some of the learned behaviors in all five tasks. The trainer had to provide feedback to teach the TAMER agent again after it learned a new mission.

We found that TAMER could only deal with "flat" tasks, and would fail to learn any complicated "compositional" task. During the training of the "alternate" task, for example, the TAMER agent could learn to navigate to the ball, but when it was trained to move to the cylinder, it forgot the learned policy about the ball and did not know what to do when seeing the ball after training.

Same problem happened in the task of "cylinder navigation", that TAMER could not learn from lure training. This is mainly because the history that TAMER could memorize is short, and could be covered by new feedback. TAMER does not support differential feedback or diminishing returns well, its model considers feedback as reward-function exemplars in which new feedback in similar contexts can change the target.

Specifically, because TAMER interprets feedback as exemplars for a reward function, different feedback in new contexts may quickly override the estimated reward-function target in similar contexts, even if the feedback was the same sign.

### **3.6 Conclusion**

In this work, we presented empirical results showing that the numeric feedback people give agents in an interactive training paradigm is influenced by the agent's current policy and argued why such policy-dependent feedback enables useful training strategies. We then introduced COACH, an algorithm that, unlike existing human-centered reinforcement-learning algorithms, converges to a local optimum when trained with policy-dependent feedback. We showed that COACH learns robustly in the face of multiple feedback strategies. A comparison experiment in simulated grid world suggested that COACH could perform well (with eligibility traces) with different types of feedback when Q - Learning and TAMER could only function with limited feedback types. Finally we showed that COACH can be used in the context of robotics with advanced training methods.



## Chapter 4

# Teaching Complex Tasks through Decomposition

This chapter addresses the problem of training a robot to carry out temporal tasks of arbitrary complexity via evaluative human feedback that can be inaccurate. A key idea explored in our work is a kind of curriculum learning—training the robot to master simple tasks and then building up to more complex tasks. We show how a training procedure, using knowledge of the formal task representation, can decompose and train any task efficiently in the size of its representation.

We further provide a set of experiments that support the claim that non-expert human trainers can decompose tasks in a way that is consistent with our theoretical results, and successfully training complex missions. We compared our algorithm with existing approaches. The results suggest that our method outperforms alternatives, especially when feedback contains mistakes.

### 4.1 Training Agent like a Dog

Interactive reinforcement learning employs human trainers as a source of feedback (Isbell et al., 2006; Thomaz and Breazeal, 2008; Knox and Stone, 2009b; Akrou, Schoenauer, and Sebag, 2011; Knox and Stone, 2013; Wirth and Furnkranz, 2013; MacGlashan et al., 2017; Christiano et al., 2017). In one view, training can be viewed as a form of communication (Ho et al., 2015; Ho et al., 2017) in which the trainer wishes to convey a target task to the learning agent and the agent wishes to infer this task and behave accordingly (Hadfield-Menell et al., 2016). The work on SABL (Loftin et al., 2014) makes this perspective explicit and we adopt it in the current work.

Classical reward functions are history-independent, but representations for temporal tasks have been proposed, often using variants of temporal logic (Bacchus, Boutilier, and Grove, 1996; Kasenberg and Scheutz, 2017). Temporal-logic representations of tasks are powerful because they can compositionally express tasks of unlimited complexity. Whereas Markov reward functions are limited to being able to express  $m^n$  distinct behaviors in an  $n$ -state,  $m$ -action environment, the number of distinct temporal tasks is countably infinite.<sup>1</sup>

In creating learning systems that can carry out a variety of behaviors for people, we can take inspiration from dog training, where a trainer can convey a seemingly unbounded collection of tasks to a dog using essentially only evaluative feedback. One significant tool in the trainer’s collection is expanding on learned tasks later in training. The site “Doggy Buddy”<sup>2</sup>, provides instructions for training 52 tricks using this kind of curricular training.

<sup>1</sup>One could argue that every possible reward function from the uncountably infinite set of reward functions does indeed represent a distinct task in that, for any pair of distinct reward functions, there exists an environment in which they induce different behaviors (Amin, Jiang, and Singh, 2017). We do not undertake a formal analysis of the relative expressibility of rewards and temporal logic in the current work.

<sup>2</sup>[www.doggiebuddy.com/topics/Trainingtopics/traintopic3.html](http://www.doggiebuddy.com/topics/Trainingtopics/traintopic3.html)

Learning to fetch a drink from the fridge, for example, builds upon first training the dog to complete a set of 8 other tricks.

Existing work for learning compositional, or logical, representations requires either an optimization procedure that can posit and recombine substructures (Koza, 1992) or a training procedure that applies feedback to separate subtasks (Rivest and Sloan, 1994). It is the latter path we follow here. We build on recent work showing that human trainers can decompose complex training tasks into more tractable curriculum structures (Peng et al., 2017; Wang et al., 2020) and examine the problem of learning a complex task specification through a series of self-contained lessons. Our approach also handles trainer error, which is prone to make traditional methods unstable (Celemin and Solar, 2019).

## 4.2 GLTL algorithm

We represent an agent’s environment as a Markov decision process (MDP). In place of standard reward functions, objectives or *tasks* are represented by Geometric linear temporal logic (GLTL) formulas (Wang et al., 2020).

That is, a task  $\Phi$  has a corresponding optimal policy  $\pi_\Phi$ , possibly non-Markovian, that results in the agent moving through the state space in a way that maximizes the probability of satisfying  $\Phi$ . A *mission* is a special task  $\Phi^*$  whose execution is the overall goal of the training process.

The problem we study is that of constructing an agent that is able to learn the desired behavior  $\pi_{\Phi^*}$  efficiently via evaluative feedback from the trainer. We write  $f_{\Phi,t} = 1$  if  $a_t \in \pi_\Phi(s_t)$ , 0 otherwise to capture the feedback expected from a trainer of task  $\Phi$  if action  $a_t$  is taken by the agent in the state  $s_t$  visited at time  $t$ .

To separate the problem of learning the MDP  $M$  from the problem of learning the desired behavior, we assume  $M$  is known to both the agent and the trainer. We measure inefficiency in learning by counting the number of times the agent takes an action that is inconsistent with  $\pi_{\Phi^*}$ .

To help the agent learn the correct task, a trainer gives either positive or negative feedback for each agent action. (We disallow neutral feedback or non-feedback at the first part of this work, then allow any feedback in later study.)

The trainer should respond with positive feedback if the agent’s actions are consistent with the desired behavior and with negative feedback otherwise. Interactions take place in *rounds*, signaled by the trainer to the agent, in which the task being taught changes from round to round.

We make two key assumptions of trainers:

- The majority of the evaluative feedback from the human trainers are accurate;
- They can select tasks to teach at each round such that tasks are either one of a relatively small set of basic tasks or a relatively simple transformation of previously learned tasks where the final round’s task is the mission  $\Phi^*$ .

We show that these assumptions are sufficient to learn arbitrarily complex missions in theory and also that users can carry out this curriculum-style training process successfully with an implemented agent and minimal prior instruction.

**Algorithm 2** GLTL algorithm

---

**Input:** basic propositions  $K_0$ , templates  $\tau$   
Initialize  $H_0 \leftarrow \tau(K_0)$ ,  $i \leftarrow 0$   
**while** trainer has not finished mission **do**  
     $L_i \leftarrow \text{LEARNTASK}(H_i)$   
     $K_{i+1} \leftarrow K_i \cup L_i$   
     $H_{i+1} \leftarrow \tau(K_{i+1})$   
     $i \leftarrow i + 1$   
**end while**  
**return**  $L_{i-1}$  as learned mission

**function** LEARNTASK( $X$ )  
     $t \leftarrow 0$   
    restarts  $\leftarrow 0$   
     $L \leftarrow X$   
    **for**  $\phi \in X$  **do**  
         $r_\phi \leftarrow 0$ , initialize formula strike counter  
    **end for**  
    choose starting state  $s_t$   
    **while** task has not ended **do**  
        observe current state  $s_t$   
        **for all**  $a \in A$  **do**  
             $c_a = \#$  formula in  $L$  with  $a$  optimal in  $s_t$   
        **end for**  
        execute  $a_t = \arg \min_a |c_a - |L||/2|$   
        **if** trainer gives feedback  $f_t$  **then**  
            **for all**  $\phi \in X$  **do**  
                **if**  $f_t \neq f_{\phi,t}$  **then**  
                     $r_\phi \leftarrow r_\phi + 1$   
                **end if**  
                 $L = \{\phi \in X | r_\phi = \min_{\phi \in X} r_j\}$   
            **end for**  
        **else if** trainer attempts to end task **then**  
            **if**  $|L| = 1$  or restarts  $\geq 10$  **then**  
                **break**  
            **else**  
                restarts  $\leftarrow$  restarts + 1  
                choose starting state  $s_{t+1}$   
            **end if**  
        **end if**  
         $t \leftarrow t + 1$   
    **end while**  
    **return**  $L$  (at most 2, selected at random)  
**end function**

---

We propose an iterative algorithm (Algorithm 2) that learns missions effectively and efficiently over a series of rounds,  $i = 0, 1, \dots, k$ .

Before the start of the first round, the agent is given  $K_0$ , a set of logical propositions in the domain. The set of initial hypotheses  $H_0$  is generated by applying the transformations  $\tau$  to  $K_0$ .

After the training for round  $i$ , the agent identifies a set  $L_i \subseteq H_i$  of learned tasks and sets  $K_{i+1} = K_i \cup L_i$ . In each subsequent round  $i + 1$ , the agent takes the set  $K_{i+1}$  and templates  $\tau$  and generates  $H_{i+1} = \tau(K_{i+1})$  from them to form new hypotheses for learning. This process continues until the trainer successfully conveys the mission  $\Phi^*$ .

At round  $i$ , the trainer guides the agent to learn  $\Phi_i$  by giving positive feedback when the agent's actions align with  $\Phi_i$  and negative feedback otherwise. During training (LEARNTASK), the agent takes actions that elicit the most discriminative feedback possible from the trainer. Specifically, the agent takes an action such that it can rule out as close as possible to 50% of the policies currently under consideration.

Given perfect feedback, whenever the agent sees feedback inconsistent with a given hypothesis, it can eliminate that hypothesis from consideration for the rest of the round. To be robust to occasional trainer errors, however, we instead have the agent allocate “strikes” to hypotheses that disagree with the feedback. When a round is ended, the agent is left with  $L$ , the set of hypotheses that had the fewest strikes against them.

In our experiments, we direct the trainer to continue training if more than one hypothesis is in  $L$ . However, after 10 restarts, the algorithm simply returns two hypotheses selected at random from  $L$ .

**Theorem 1.** *Consider a set of tasks  $X$ , where every  $x \in X$  has a formula of length at most  $d$  and can be distinguished from the rest of the tasks in  $X$  using a trajectory of length  $m$ . Given that  $|X|$  is polynomial in  $d$  and provided with evaluative feedback with at most  $n$  errors, a learning agent can successfully identify the target task with a number of interactions that is polynomial in  $m$ ,  $n$ , and  $d$ .*

Theorem 1 shows us that, if we know a trainer will make only at most  $n$  mistakes, we can ensure that we find the single correct hypothesis. It would be difficult, if even possible, to know the maximum number of mistakes a particular human trainer might make, so this assumption is a strong one.

We can calculate a bound on the error rate of the trainer that the naïve algorithm in the proof will tolerate. Since we can err up to  $n$  times safely on each of the tests with  $(n + 1)m$  possible feedback signals, our acceptable error rate  $\rho$  is bounded by

$$\rho = \frac{n}{(2n + 1)m} < \frac{1}{2m}.$$

Further, Algorithm 2 follows the same logic as the algorithm used to prove Theorem 1, except it more efficiently tests hypotheses by using feedback to gain information about multiple hypotheses at once. Requiring fewer feedback signals allows for the algorithm to be tolerant of higher rates of trainer error, in practice.

Similar to standard Boolean logic, GLTL formulas can be represented as syntax trees with nodes for the additional LTL operators. We partition the space of all GLTL formulas into two classes, *temporal* and *atemporal*.

We define a temporal GLTL formula to be one in which, on every root-to-leaf path in the tree representation, there exists at least one temporal operator ( $\diamond$ ,  $\square$ ,  $\mathcal{U}$ ). Conversely, an atemporal GLTL formula is one in which there exists at least one root-to-node path in the syntax tree that does not contain a temporal operator. Any task that is defined by a temporal formula is referred to as a temporal task and any task defined by an atemporal formula is referred to as an atemporal task.

Known Formula		Derived Formula
<b>Atemporal transformations</b> (X and Y are not necessarily temporal formulas)		
1.	X	$\diamond X$
2.	X	$\square X$
3.	X, Y	$XUY$
<b>Temporal transformations</b> (X, Y, are temporal formulas, x and y are not necessarily temporal formulas)		
4.	$X = \diamond x$	$\diamond \neg x$
5a.	$X = \diamond x, Y = \diamond y$	$\diamond(x \vee y)$
5b.		$\diamond(x \wedge y)$
6.	$X = \square x$	$\square \neg x$
7a.	$X = \square x, Y = \square y$	$\square(x \vee y)$
7b.		$\square(x \wedge y)$
8.	$X = \square x, Y = \diamond y$	$xUY$
9.	X	$\neg X$
10.	X, Y	$X \wedge Y$
11.	X, Y	$X \vee Y$
12.	X	$\square \neg X$
13.	$X = \diamond x, Y$	$\diamond(x \wedge Y)$
<b>Specialized transformations</b> (X, Y, Z, x, y need not be temporal)		
14.	Z, XUY	$XU(Y \wedge Z)$
15.	$X = \diamond x$	$\diamond \square x$
16.	$X = \square x$	$\square \diamond x$
17.	$\diamond(X \wedge \diamond y)$	$\diamond(X \wedge \diamond \square y)$

FIGURE 4.1: Templates used in constructing temporal formulas.

Table 4.1 lists the templates we use in our algorithm. The templates were chosen by inferring what kinds of transformations participants seemed to be expecting in the context of a pilot study, but they were then significantly modified to both ensure all temporal tasks could be constructed and strike a balance between coverage and complexity.

The significance of temporal tasks is that they cannot be shown to be unsatisfied without at least a single temporal step, meaning that a trainer will have an opportunity to provide feedback.

Therefore, to show that a mission can be trained by our multi-round curricular algorithm, it must be a temporal task *and* it must be able to be constructed via the templates we define where every task en route to the mission is formulated as a temporal task.

**Lemma 2.** *Any temporal formula can be built up via application of these transformations starting from basic tasks that include simple temporal formulas of the domain's propositions. All of the intermediate formulas in the construction are themselves temporal.*

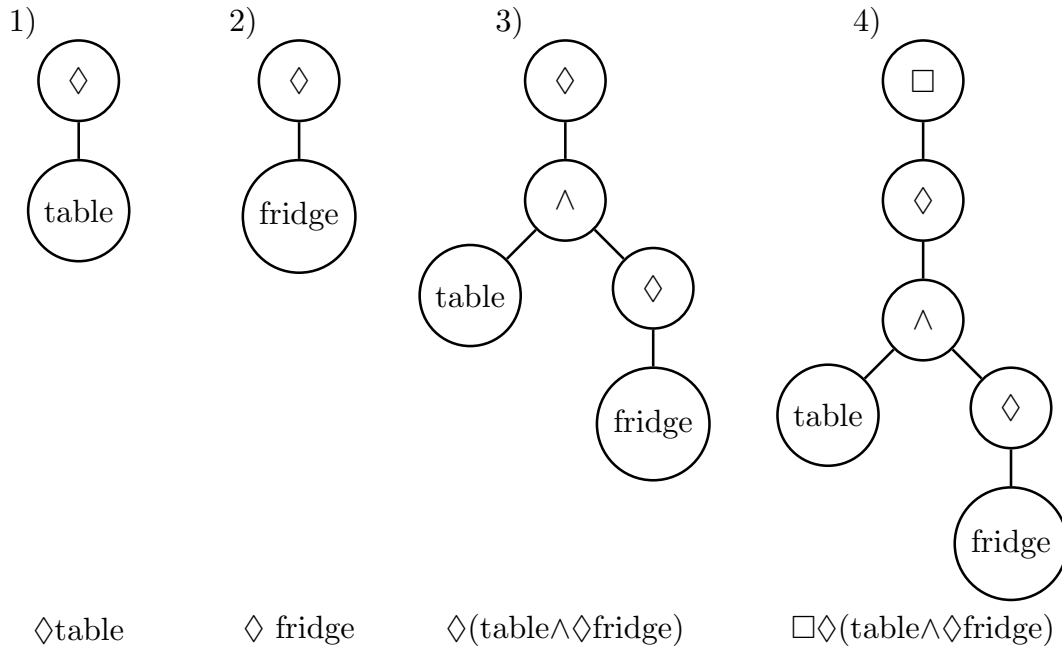


FIGURE 4.2: The training procedure for task  $\square\Diamond(\text{table} \wedge \Diamond\text{fridge})$ . Steps 1) and 2) use template 1 from Figure 4.1. Step 3) uses template 13. The final step, 4) uses template 2.

An intuitive description of the proof sketch for Lemma 2 is the following. Whenever we're training at a node that is an atemporal operator, we know, by definition, that both children must be temporal. So, we could have trained (bottom up) to this point without any issues via induction. When the current node is temporal, it is possible that one or both of its subtrees is atemporal. In that case, we can show that we can propagate the temporal operator down to the subtrees. If we train according to this new tree structure, the atemporal subtrees become temporal and therefore trainable. This training strategy is illustrated in Figure 4.2.

**Theorem 3.** *If a trainer can decompose a mission into tasks satisfying the lemma and can provide evaluative feedback with low error rate, they can train the algorithm to learn the temporal LTL task in time polynomial in the size of the formula with high probability.*

Theorem 3 follows from a combined application of Theorem 1 and Lemma 2. We show that the training procedure remains feasible with respect to the size of the formula for the overall mission.

In short, Theorem 1 shows that any temporal task we consider can be correctly learned via imperfect feedback. Lemma 2 explains that any temporal mission can be successfully decomposed and trained as a series of these smaller tasks discussed in Lemma 2.

Finally, Theorem 3 proves that if a trainer is able to decompose tasks according to Lemma 2 and can provide feedback with low error as in Theorem 1, then Algorithm 2 can efficiently learn GLTL missions.

Having shown theoretically that our learning algorithm can be efficiently taught any temporal mission formula by an idealized, low-error trainer, we follow up by showing that real world users can approximate the idealized trainer sufficiently closely to convey complex tasks to the agent.

## 4.3 Experiments and results

We carried out a set of five experiments with the goal of determining whether people could reason about the skills needed to train an agent to carry out complex missions.

### 4.3.1 Mission Decomposition Study

In the first study, participants were asked to imagine training a hypothetical home robot. Specifically, the instructions said: *To teach “go get a glass of water from the kitchen without going through the living room,” would you teach the expert task directly or teach beginner and intermediate tasks first? What beginner/intermediate tasks would you teach and in what order?*

The five given missions were:

- Go to your charger without colliding with either the couch or the chairs.
- Go to the spill someone left on the floor and then go to the cabinet and stay there.
- Go back and forth patrolling between the bedroom door and bedroom window.
- Go to the water dispenser and back to the couch without getting in the way of the TV.
- Go from the kitchen to the bedroom. It is okay to go through the living room, but only after muting any beeping sounds.

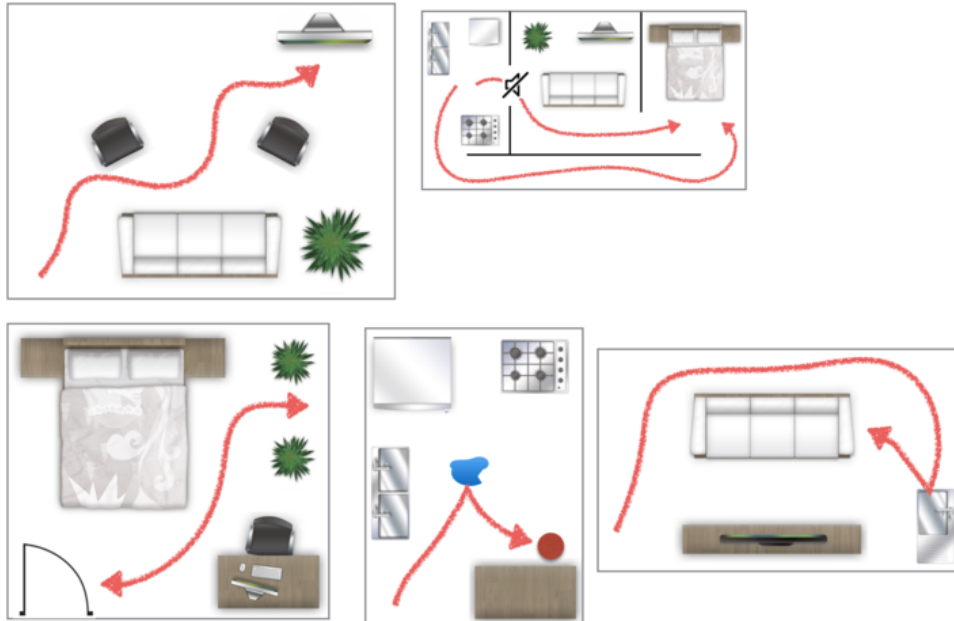


FIGURE 4.3: Floor plan images to help ground the questions.

Some pilot participants found our mission scenarios too abstract, so we included accompanying floor plan images (Figure 4.3) to help ground the questions. Some participants made reference to the details of these diagrams, but we interpreted their instructions in as generic terms as we could. The order in which the missions were presented to the participant was randomized.

Participants were recruited from Amazon Mechanical Turk (AMT) and were filtered based upon their Human Intelligence Test Approval Rate (greater than 95%) and Number of Human Intelligence Tests Approved (greater than 1000). The study awarded \$0.80 to participants.

### 4.3.2 Mission Decomposition Study results

The mission decomposition study included 20 participants, each of whom provided skills for each of the five target missions. For a total of 75 proposed training sequences. Training sequences averaged 3.4 skills in length.

Two well trained researchers from our team served as coders, translating each skill description into GLTL independently, then conferred with a third coder to reach consensus on the translations. Descriptions that did not appear to describe a trainable skill for the situation were tagged as “uninterpretable”. On average, participants listed 1.2 uninterpretable skills per target mission. As an example, one participant broke down Mission 3 into:

1. go to the bedroom door;
2. go to the bedroom window;
3. go to the door, then go to the window, then repeat.

We translated the skills into:



1.  $\diamond$ door;
2.  $\diamond$ window;
3.  $\square\diamond(\text{door} \wedge \diamond\text{window})$ .

In contrast, another participant broke down the same mission into:

1. learn the geography of the home;
2. navigate a clear pat;
3. return and repeat.

which we classified as uninterpretable.

### 4.3.3 Recomposition Study

A useful aspect of the first study was that participants had the freedom to decompose missions however they felt was appropriate. A shortcoming, though, is that they received no feedback as to whether the decomposition they proposed was viable.

Based upon the training patterns observed in the mission decomposition study, the second study presented participants with the same basic problem of decomposing a complex mission into skills, and we had them select the skills from a list of common responses that participants generated in Study 1. Between 5 and 8 skills were listed for each mission.

We wanted to get a sense of whether they could modulate their design when they were headed down a problematic path. Each time participants in the recomposition study selected a skill, they were given feedback as to whether the previously selected skills provided the necessary foundation for the agent to learn the selected mission based upon our established curricular learning templates 4.1 and the GLTL translations from the previous study.

The missions and their GLTL interpretations were:

1. Go to your charger without colliding with either the couch or the chairs.  
The GLTL formula is:  $\neg(\text{couch} \vee \text{chair}) \mathcal{U}\text{charger}$ .
2. Go from the kitchen to the bedroom. It is okay to go through the living room, but only after muting any beeping sounds.  
The GLTL formula is:  $\text{kitchen} \wedge \diamond\text{bedroom} \wedge \square(\text{livingroom} \implies \text{muted})$ .
3. Go back and forth patrolling between the bedroom door and bedroom window.  
The GLTL formula is:  $\square\diamond(\text{door} \wedge \diamond\text{window})$ .
4. Go to the spill someone left on the floor and then go to the cabinet and stay there.  
The GLTL formula is:  $\diamond\text{spill} \wedge \diamond\square\text{cabinet}$ .
5. Go to the water dispenser and back to the couch without getting in the way of the TV.  
The GLTL formula is:  $(\neg\text{TV}) \mathcal{U}(\text{dispenser} \wedge \diamond\text{couch})$ .

Missions were given in a random order in this study as well.

#### 4.3.4 Recomposition Study results

20 participants were recruited using the same procedure as the mission decomposition study. Participants averaged 7.6 skill attempts to train each of the missions, where 3.0 of these skills were extra steps that were either erroneous or unnecessary to train the target mission.

We analyzed how many extra steps participants took as a function of the mission ordering. Comparing the 5<sup>th</sup> mission they solved to the first mission they solved, the number of extra steps decreased by an average of 1.55. This value is statistically significantly different from 0 ( $p < .05$ ) by an unpaired t test, suggesting that participants experienced improvement in their ability to select appropriate skills, even over this short experience.

Note that the number of skills participants attempted per target mission was 7.6, which is longer than the average list of skills (6.6) and could therefore be interpreted as the participants randomly hunting and pecking to create a valid training ordering. As a baseline, we computed the average training ordering required by truly random selection, which was 21.3. Thus, participants were very effective at skill selection compared to a random approach.

We define a decomposition as *sufficient* if the temporal formula for the mission can be built up via application of temporal transformations starting from the set of basic tasks. In the experiment, we determined the sufficiency based upon our templates (Table 4.1) and our GLTL translations of the participants' tasks.

An *unnecessary* task in a sufficient decomposition is one that can be removed and the decomposition remains sufficient.

A sufficient decomposition is *minimal* if none of its tasks are unnecessary.

### 4.3.5 Simulation study

The mission is: Go back and forth patrolling between the table and the fridge.

The robot has learned:

go to the table

Your goal this time is to teach the robot:

go to the fridge

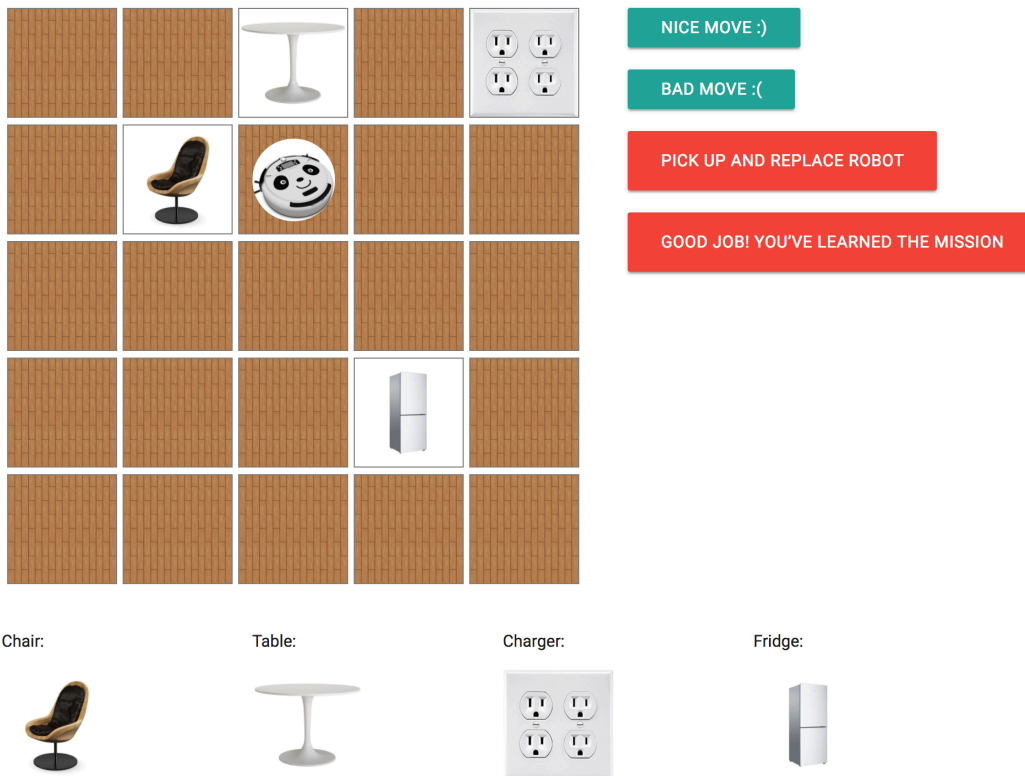


FIGURE 4.4: Experimental domain for teaching temporal tasks.

We then conducted a simulation study to compare the performance of our algorithm with TAMER and COACH. we evaluated how well each algorithm performed in a simulated  $5 \times 5$  grid world (Figure 4.4). In the grid world, there were four objects, each associated with its own atomic proposition: a table, a chair, a charger, and a fridge. Four tasks were tested:

1. (1A) Move all around, but don't bump into the chair:.  
The GLTL formula is:  $\square \neg \text{chair}$ .
2. (1B) Go directly to the table:.  
The GLTL formula is:  $\diamond \text{table}$ .
3. (1C) Don't touch the table on your way to reaching the charger:.  
The GLTL formula is:  $\neg \text{table} \mathcal{U} \text{charger}$ .
4. (1D) Start at the fridge and stay there.  
The GLTL formula is:  $\square \text{fridge}$ .

The tasks were all in the initial set of hypotheses of the GLTL algorithm set and thus could be learned directly without a curriculum strategy according to Table 4.1, Algorithm 2.

There were two types of trainers we simulated in this experiment:

1. an ideal trainer who only gives correct feedback;
2. a simulated non-expert trainer who gives feedback with an error rate of  $\theta$ .

For each type of trainer, we ran 10 rounds of training for each algorithm. During each round, the agent started from a random grid and learned from feedback given by the trainer until the task was complete, then chose another random position to start the next episode.

We measured how well the learner performed by the percentage of positive feedback its trajectory received from an ideal trainer. A *perfect* episode is one with a value of 100%, which means that all the actions that the agent has taken matched the optimal policy of the target task. Once the agent gets three perfect episodes in a row, the round is finished.

We counted the median number of episodes the agent had taken before the first perfect episode, the median number of feedback per episode, and the median number of the total feedback per round. For task 1D, the agent always started from the fridge because the task cannot be completed successfully otherwise.

Since the experimental tasks was temporal rather than state-based in nature, we augmented the state space for TAMER and COACH so that they could learn the target behavior. The state consisted of the current position of the robot along with the order that the landmark object were visited up to that point. So, if the robot had visited the charger and the table, in that order, the state space would be augmented with “1: charger, 2: table”.

Our algorithm took a very small number of episodes to learn the tasks.

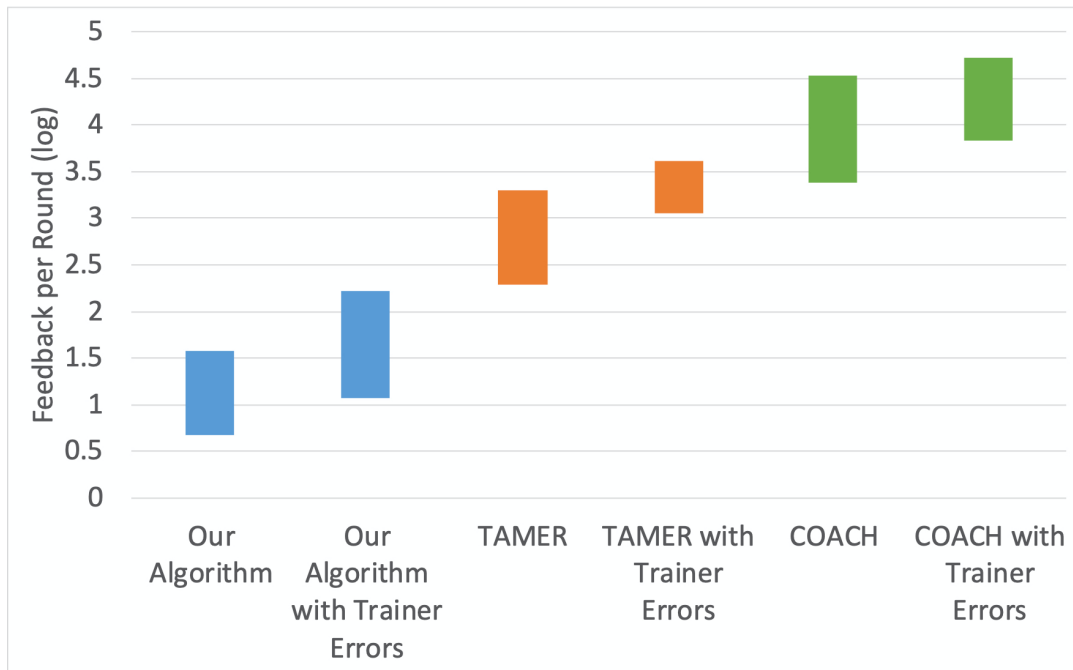


FIGURE 4.5: The number of training feedback needed for our algorithm, TAMER, and COACH across tasks:  $\square$ —chair;  $\diamond$ —table;  $\neg$ —table  $\cup$  charger; and  $\square$ —fridge.

Figure 4.5 presents the results of the average number in log scale of feedback it took for each of the algorithm to learn the tasks in two scenarios—error-free feedback and feedback

with  $\theta = 0.327$  errors. This value was chosen to align the simulation with our observation that 32.7% of feedback signals were observed to be in error in one of our user studies. For each algorithm, we show the range of training times over the four tasks.

The results show that our algorithm needs significantly fewer episodes to learn the tasks—the simulated trainer gave much more feedback per round to TAMER and COACH than to our algorithm. The performance difference between our algorithm and the other two became larger when the trainer made mistakes. (Note the log scale on the plot.)

We further tested the algorithms with a group of complicated missions:

1. (2A) Go to the table and then go to the charger and stay there.  
The GLTL formula is:  $\diamond(\text{table} \wedge \diamond\Box\text{charger})$ .
2. (2B) Go to your charger without colliding with either the chair or the table.  
The GLTL formula is:  $\neg(\text{chair} \vee \text{table}) \mathcal{U}\text{charger}$ .
3. (2C) Go to the table and then go to the fridge.  
The GLTL formula is:  $\diamond(\text{table} \wedge \diamond\text{fridge})$ .
4. (2D) Go to the charger and then go to the chair without running into the table along the way.  
The GLTL formula is:  $(\neg\text{table}) \mathcal{U}(\text{charger} \wedge \diamond\text{chair})$ .
5. (2E) Go back and forth patrolling between the table and the fridge.  
The GLTL formula is:  $\Box\diamond(\text{table} \wedge \diamond\text{fridge})$ .

Neither TAMER nor COACH could learn any of them. Our algorithm, in contrast, successfully learned the missions from decomposition.

#### 4.3.6 User Study: Learning Basic Tasks from Non-expert Human Trainers

We then carried out two user studies to determine whether non-expert human trainers could

1. train a learning agent running our algorithm to execute basic tasks;
2. train a learning agent end to end on complex missions by decomposing it into smaller tasks and providing evaluative feedback.

Participants for each study were recruited via Amazon Mechanical Turk (AMT) as previous online user studies.

In the user study where the agent learns basic tasks from non-expert human trainers, participants were presented with a simulated robot (the learning agent) in a  $5 \times 5$  grid world (Figure 4.4). A brief introduction of the user study environment can be found here: <https://youtu.be/40uLW10UFzk>.

During each round, participants were given a basic task that they needed to train the robot to execute via positive and negative feedback. Each participant was asked to complete four rounds, each with a different and independent task. The tasks were same as the four basic tasks in the simulated environment, and were given one by one in a random order to the participants.

At the beginning of each round, the participant was asked to place the robot by clicking on one grid cell in the map. The robot would then choose an action (moving up, right, down, left, or stay) based on our scheme. After taking an action, the agent would wait for the participant to give feedback. The participant could choose to give positive feedback by clicking on “NICE MOVE :)”, or negative feedback by clicking on “BAD MOVE :(”. Our algorithm would then learn from the feedback, ruling out formulas that did not match the feedback and choosing from the remaining formulas to make its next action selection.

During training, the participant could change the agent’s location by clicking “PICK UP AND REPLACE ROBOT” and choose another grid cell to continue. Once the participant was satisfied with the performance of the agent, he or she could click “GOOD JOB! YOU’VE LEARNED THE MISSION” to end the training. At the end of training, the remaining set of formulas, as derived by the algorithm, was considered to be the *learned formula set*.

We instructed participants:

1. to give positive feedback if he or she believed that the most recent action was consistent with the target task, and provide negative feedback otherwise;
2. to end the training when he or she believed that the agent had learned the task.

### 4.3.7 User study Results: Learning Basic Tasks from Non-expert Human Trainers

To measure task success, we evaluated whether the learned formula matched the target formula using a *similarity* score defined over the range  $[0, 1]$ . Consider a ground truth formula  $A$  of which the induced policy is  $\pi_A$ , and a learned formula  $B$  with policy  $\pi_B$ . For  $\pi_B$ , we create a set of trajectories  $J_B$  by starting the agent from each of the twenty-five positions on the grid and executing twenty random trajectories of  $\pi_B$ . Thus,  $J_B$  contains five hundred trajectories in total.

The *similarity* of  $B$  to  $A$  is calculated by counting the percentage of positive feedback of all trajectories in  $J_B$  from an ideal trainer trying to teach  $A$ :

$$\text{similarity}(B, A) = \sum_{J_B} \frac{f_{\text{Positive}}}{f}$$

where  $f_{\text{Positive}}$  is the total number of positive feedback of each trajectory,  $f$  is the total number of feedback of each trajectory. It can be thought of as telling us how happy someone would be when they were wanting to train formula  $A$  and the robot exhibits behavior from formula  $B$ .

If the agent learns more than one formula, then we conservatively report the minimum similarity for all learned formulas to the target formula.

In the  $5 \times 5$  grid world, the similarity between two randomly generated formulas is 0.16 on average. To get a sense of the meaning of the scores, here are some examples, suppose there are four formulas:

- formula  $A$  is  $\square$ fridge.
- formula  $B$  is  $\diamond$ charger.
- formula  $C$  is  $\diamond$ fridge.
- formula  $D$  is  $(\neg\text{table})\mathcal{U}$ charger.

We get:

- The similarity of formula  $A$  to  $B$  is 0—they send the agent in very different directions.
- The similarity of formula  $B$  to  $C$  is 0.35, because both draw the agent to the right side of the grid.
- The similarity of formula  $D$  to  $B$  is 0.98 because they produce nearly the same action in all states.

A training round was considered to be successful if the learned formulas were above a specific similarity threshold to the mission. The success rate is the percentage of rounds that were successful.

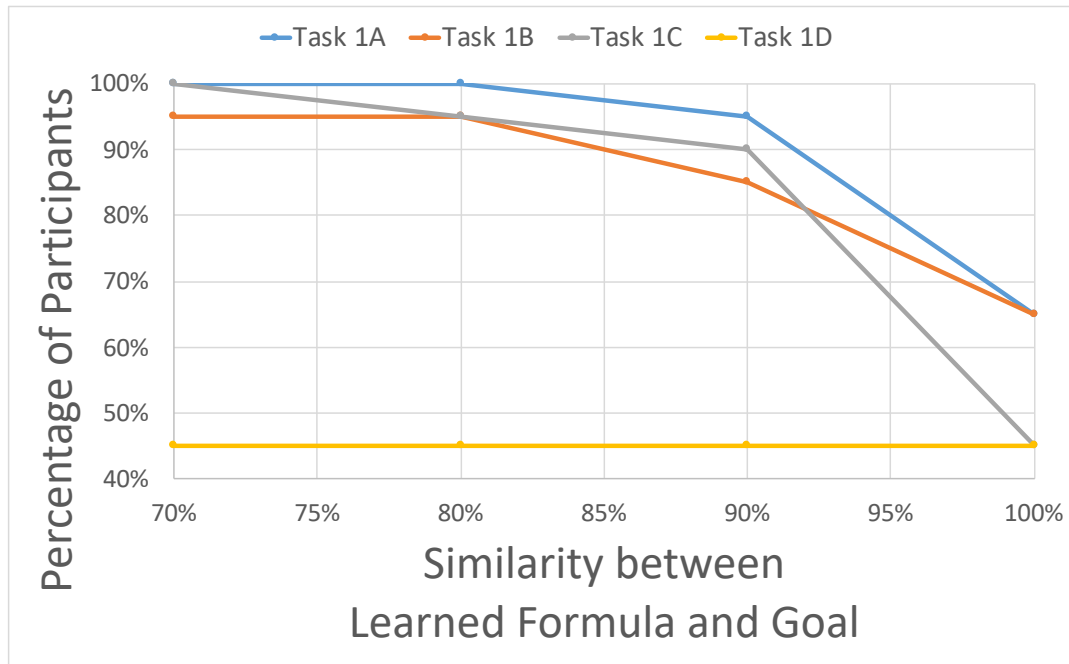


FIGURE 4.6: Task success rate in user study: Learning Basic Tasks from Non-expert Human Trainers.

In this study, there were 80 training rounds (20 participants and 4 rounds each). Figure 4.6 shows the success rate for different similarity thresholds for each target task. The horizontal axis shows the similarity between the learned and target tasks. The vertical axis shows how many of the participants reached the similarity score in each task.

A higher similarity, e.g., 0.9 or greater, means that in 90% or more of the situations the two formulas expect the same feedback for the same action. If the similarity between two formulas is 1, then they can be treated as the same formula. As Figure 4.6 suggests, that 63 out of 80 missions (78.75%) were finished with a similarity of at least 0.9.

The results support our hypothesis that the majority of people can teach simple temporal tasks, with the exception of task `fridge`. An examination of the logs suggested that participants were unsure what feedback to give for this task when the robot did not start at the fridge. Some gave positive feedback to the robot for approaching the fridge, causing the learning algorithm to fail.

We found that the success rate for a participant was strongly correlated with the number of replacements used in training. At a similarity threshold of 0.9, trainers who successfully trained 3 or more rounds (among all 4 rounds) clicked “PICK UP AND REPLACE ROBOT” 21 times on average; in comparison, trainers who successfully trained only 2 or fewer rounds used replacement 9 times on average.

This finding was likely a result of the fact that replacements expanded the size of the effective training set for the learning agent. Moreover, a good replacement (one that can efficiently help the agent distinguish between the remaining formulas) can speed up the training and improved performance.

Based on this finding, we added functionality to our interface so that, whenever multiple formulas remained when the participant requested to end training, the agent would automatically move itself to a new start position to help disambiguate the remaining formulas. Note

that the learning agent was not aware of the actual target of training, just that there was residual ambiguity in what it had learned. This form of active learning greatly simplified the training process for participants.

#### **4.3.8 User study: Learning Complex Missions via Decomposition**

In this user study, we had participants train an agent to carry out complex missions by decomposing each mission into a curriculum of simpler tasks and training the tasks one by one.

As in the previous user study where the non-expert participants were asked to train basic tasks, we recruited 20 participants from AMT. The participants were presented with the same  $5 \times 5$  grid world and the simulated robot (Figure 4.4). Participants received the same instruction as in the previous study regarding how to give feedback and when to end a training round. The robot chose actions and waited for feedback.

Based on the result of the previous user study, an update was made to the interface and the learning algorithm: at the end of each round, if there were more than two formulas in the learned formula set, the robot would automatically move to a new start position. A training round could only be finished if the number of remaining potential formulas in  $H$  was one or two.

Unlike the previous user study, we presented each participant with three independent complex missions chosen randomly from the group of five complicated missions in the simulated environment. After showing them a tutorial, we asked them to first decompose each mission into a sequence of simple tasks, the last task being the mission itself, then train one task each time until the final mission was learned.

The tasks were trained consecutively in rounds. Within each round, the training process is executed as in previous study. The participants can revise the sequence of tasks after each round. The English description of each decomposed task was recorded and later translated by the authors into GLTL.

The learned formula was derived in the same way as in the previous study. For the last task in each mission, the target task is the mission itself and the learned formula represents the learned formula for the entire mission.



### 4.3.9 Results of User study: Learning Complex Missions via Decomposition

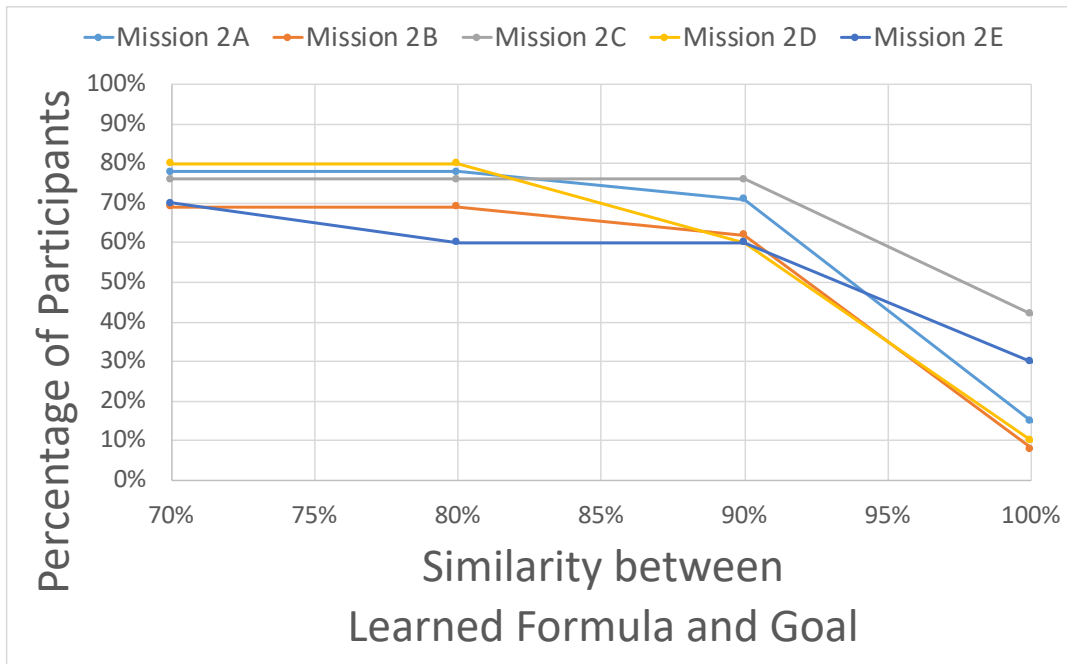


FIGURE 4.7: Mission success rate in user study: Learning Complex Missions via Decomposition.

Figure 4.7 shows the success rate for different similarity thresholds for each mission. Figure 4.8 shows the success rate for different similarity thresholds for the decomposed tasks of each mission. The two figures were generated in the same way as Figure 4.6.

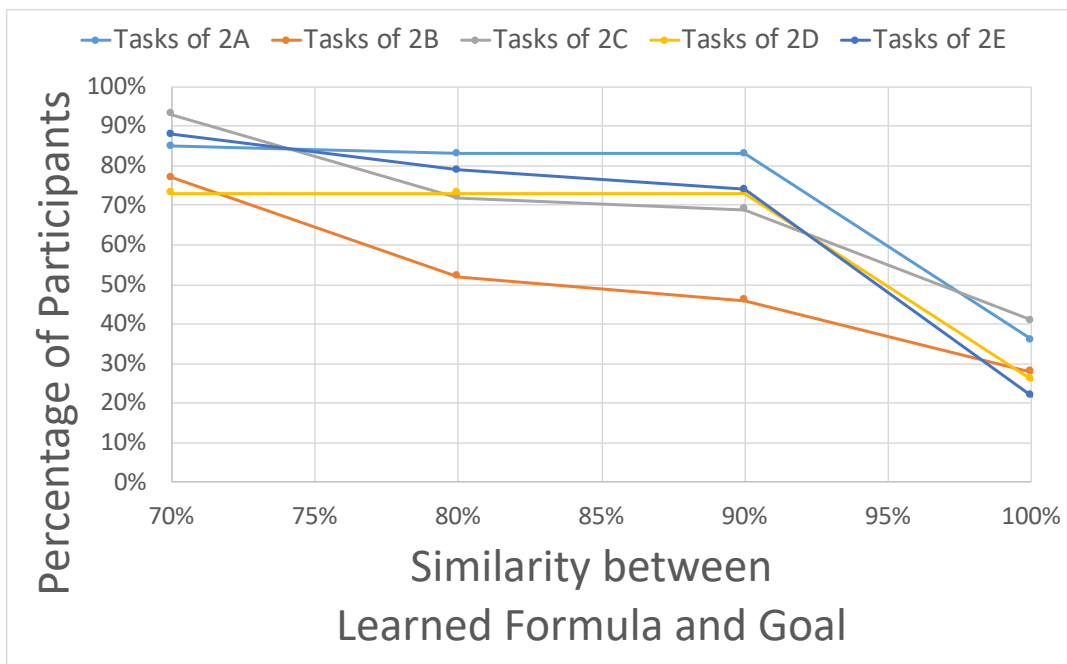


FIGURE 4.8: Task success rate in user study: Learning Complex Missions via Decomposition.

The study result supported our hypothesis that the majority of human trainers can teach complex temporal missions via our algorithm. For all five missions, more than 60% of participants successfully taught the agent the formulas (similarity threshold 0.9), which means that for any trajectory of the learned formulas, 90% or more behavior are exactly the same as the target formula.

Using a slightly weaker criterion, we see that 90% training sessions resulted in a similarity of 0.7 or higher. Note that similarity dropped as the complexity level of goal mission (measured in terms of the number of operators) increased.

Figure 4.2 shows the most common sequence of formulas trained by participants for Formula 2E. All missions required a multi-step decomposition, and participants produced a formula with similarity over 0.8 with the target formula in more than half of all attempts.

Among all the decompositions, 63.4% were sufficient and 50.1% were minimal. When the decomposition was insufficient or included unnecessary tasks, the mission could still be learned successfully with reasonably high probability as shown in Figure 4.7 and Figure 4.8.

In 25 of the 59 runs, the similarity of the learned mission to the target mission was higher than the similarity of at least one of the learned intermediate tasks to the user's attempted intermediate task. For Missions 2D and 2E, the agent was able to learn the mission even when at least one of the decomposed tasks failed.

Among the five formulas, participants achieved the best performance on Mission 2C. All but one (11 of 12) participants divided the mission into

1. go to the table;
2. go to the fridge;
3. go to the table and then go to the fridge.

and succeeded. The participant who failed neglected to train the second skill, which left the agent without the target mission in its hypothesis set.

For Mission 2D, the median number of skills created was 2.5 and 7 out of 8 (87.5%) participants achieved a similarity of 0.8 or higher. The one who failed proposed the sequence:

1. go to the table;
2. go to the charger;
3. go to the charger and then go to the chair without running into the table along the way.

There was too big of a jump from skill 2 to skill 3. Two participants realized the agent was not able to learn a skill they proposed, and so revised the skill and completed the mission successfully.

We examined how often trainer feedback is in error given our LTL translations of the tasks. Across all training sessions, 4739 of 14505 (32.7%) feedback signals were in error. While this error rate was higher than expected, it yields a promising success rate (at a similarity threshold of 0.9), which was at least 60% across all the five missions. The multi-round training procedure allows participants to successfully train tasks even in the face of high error rates for the individual feedback signals.

## 4.4 Conclusion

In this chapter, we addressed the problem of learning complex tasks via evaluative feedback through a sequence of self-contained lessons. We provided theoretical results showing the effectiveness of our approach, then followed up with experiments in simulation.

The results show that our algorithm outperformed existing approaches—TAMER and COACH—by taking significantly fewer training episodes and feedback signals to finish training. In a series of user studies, we invited non-expert participants to decompose complex missions, recompose tasks, train simple tasks, and asked them to train complex missions. The results suggest that, in spite of people making mistakes in feedback and decomposition, they were very often able to convey a formula very similar to the target mission.



## Chapter 5

# Interactive Learning from Human Feedback and Natural Language

This chapter introduces a novel approach that enables non-expert human trainers to teach a robot complex missions through giving natural language command and feedback.

### 5.1 Why Natural Language?

In Chapter 4, I've made it possible for non-expert people to teach their robots by giving positive and negative feedback. A natural next step is to bring the interaction to a deeper level: speaking. Most people give rich natural language commands to their dogs instead of simply saying just 'good!' or 'bad!'. If dogs can "understand" the meaning of complex language commands and perform the desired behavior, e.g., "go get the leash and send it to Mom", it may be possible to teach robots in a similar way.

The capability of learning from human natural language through interaction opens the door to adaptive personalized AI, which could adapt to the behavior and habit of any human trainer without large amount of labeled data. Such interactions are not limited to specific languages, English, Mandarin, Korean, Spanish, can work in the same way.

### 5.2 Background

Natural Language Processing (NLP) is an important research area of machine learning and human-computer interaction. Semantic parsing, as the core of Natural Language Understanding (NLU), focuses on the problem of mapping from natural language utterances to other representations (Kamath and Das, 2018), which involves prediction of inherently tree-ish structured objects.

In early stage, researchers used rule-based systems in narrow domains where statistical models on input-output mappings could work. Such mapping usually relies on labeled data, which has always been expensive. SAVVY (Johnson, 1984), for example, is built on pattern matching. LUNAR (Woods, 1973) is a syntax-based system that generates rule-based tree structure mapping to an underlying database query language. (Zelle and Mooney, 1996) trained their CHILL model on a corpus of sentence-database query pairs on the basis of inductive logic programming. In Zettlemoyer and Collins' work (Zettlemoyer and Collins, 2012), they introduced lambda-calculus expressions, which contributed to many later works like GeoQuery (Davis and Meltzer, 2007).

Supervised learning approaches have been very popular in the past few decades. (Krishnamurthy and Mitchell, 2012) used syntactic and semantic information to train a versatile semantic parser with dependency parsed sentences from a web corpus, and achieved 56% recall of target form. (Berant et al., 2013) trained a semantic parser on Freebase. They used a coarse grained solution using a large question-answer knowledge base. (Pasupat and

Liang, 2015) came up with a model to answer complicated questions on semi-structured tables through supervision of question-answer pairs.

The first attempt to use unsupervised approach is done by (Poon and Domingos, 2009). They cluster tokens by types and recursively combine sub-expressions by cluster. Although their model that can map passive and active voices to the same form is robust, they can not deal with complicated questions without extra ontology matching. (Artzi and Zettlemoyer, 2011) use conversational feedback from unannotated logs to learn representation for user utterances.

Statistical machine translation techniques are introduced by (Wong and Mooney, 2006) to achieve a robust model with good performance on word ordering. They treated the parsing model as a syntax-based translation model.

Learning directly from user feedback gets more attention in the past few years. (Iyer et al., 2017) adapt neural sequence models to map utterances directly to Structured Query Language (SQL) with its full expressiveness, bypassing intermediate meaning representations. The model is then deployed online to collect user feedback for incorrect predictions. Their experiments suggest that their approach can quickly learn a semantic parser from scratch for any new target domain.

By combining context words from the ground truth with the information from predicted sequence, (Zhang et al., 2019) successfully reduced over-correction in neural machine translation. They compared generations of the predicted words in the manner of word-level and sentence-level. The empirical results showed that sentence-level predicted words had the ability of over-correction recovery and their method achieved significant improvement on several experimental datasets.

### 5.3 Sequence to Sequence Model

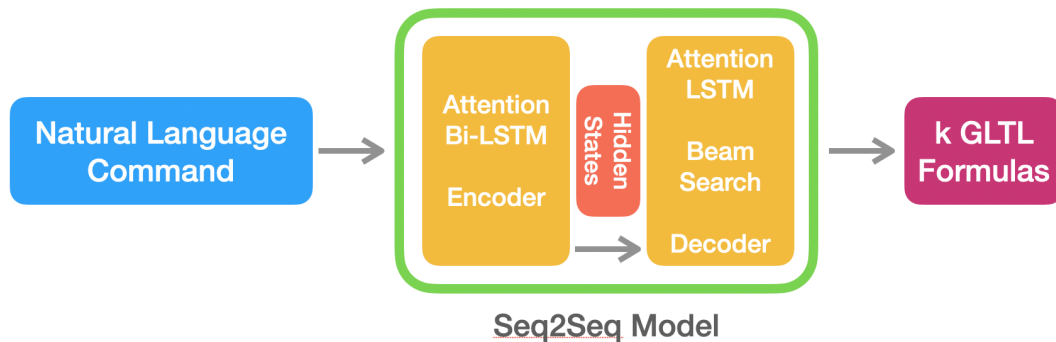


FIGURE 5.1: Sequence to Sequence Model.

Recently, a trend rises in NLP taking greater emphasis on deep neural networks (DNNs) with sequence-to-sequence (Seq2Seq) model (Sutskever, Vinyals, and Le, 2014). It has pushed the state-of-the-art performance of multiple sequential-task areas to a higher level, such as machine translation, speech recognition, syntactic parsing, image captioning, conversation, summarization, and question and answering (Sutskever, Vinyals, and Le, 2014; Kalchbrenner and Blunsom, 2013; Bahdanau, Cho, and Bengio, 2014; Cho et al., 2014b; Vaswani et al., 2017).

The shared advantages of these models is to use the encoder-decoder frameworks for end-to-end approaches. A sequence-to-sequence model is often composed of an encoder, a decoder, and an intermediate step. The structure of the Seq2Seq model is displayed in Figure 5.1.

### 5.3.1 Recurrent Neural Networks

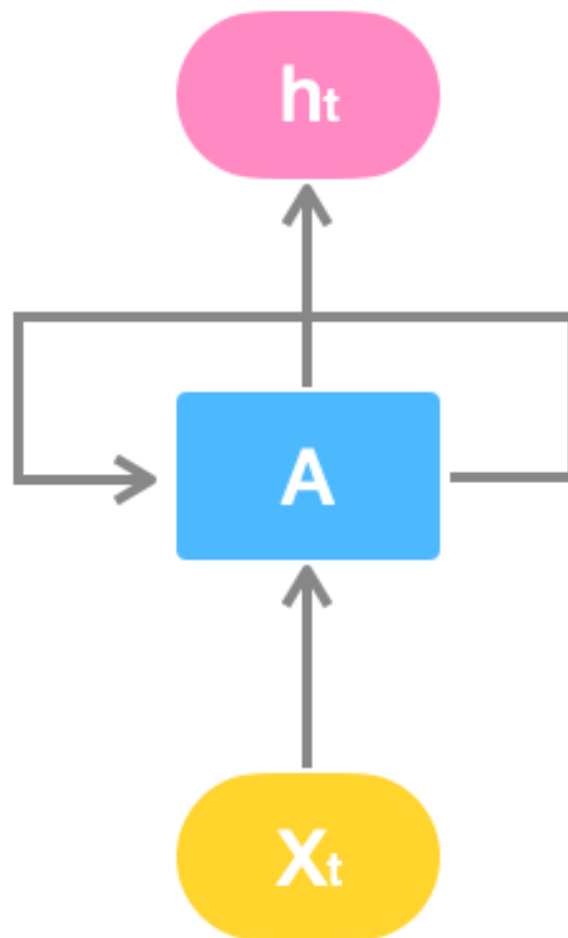


FIGURE 5.2: Recurrent Neural Networks have internal loops.

Recurrent Neural Networks (RNN), introduced by (Rumelhart, Hinton, and Williams, 1986), are considered as early foundations of the domain. RNN represent a class of neural networks where connections between nodes from a directed graph along a temporal sequence, which allows it to exhibit temporal dynamic behavior. RNN use a looping mechanism that makes them a good choice for processing sequential data of variable length. As is shown in Figure 5.2, which is a chunk of RNN. Given the input  $X_t$ ,  $A$  generates output of  $h_t$ , allowing information to be passed from one step of the network to the next one. RNN-based encoder-decoder frames have shown success in several NLP tasks like syntactic parsing (Vinyals et al., 2015).

On the other hand, simple RNN model has a major drawback, known as the vanishing gradient problem, which causes RNN to perform badly in certain scenarios. In short, at each time step during training, same weights are used to calculate the predicted next item, which is usually the predicted next words for NLP problems. The multiplication also happens in back-propagation. The further it move backwards, the bigger or smaller the error signal will become, which caused the problem of losing information. RNN has difficulty in memorising words from far away in the sequence and therefore could only rely on the recent words for prediction.

RNN also don't see hierarchy in sequence. The models alter the hidden state information every time when a new input is processed regardless of the level of significance. Moreover, because parts of the input are being computed one at a time, RNN is usually very slow, which is an important disadvantage especially when the application is used with humans. In a word, people do not want to wait for the AI to finish upgrading itself if it learns from interaction.

To solve this problem, several solutions have been proposed, among which the Long Short-Term Memory (LSTM) is considered to be the most popular model.

### 5.3.2 Long Short-Term Memory

LSTM is fundamentally a special version of RNN architecture firstly introduced by (Hochreiter and Schmidhuber, 1997). The feedback connections make it be able to not only process single data points but also a sequence of inputs. LSTM is explicitly designed to solve the long-term dependency problem. A standard LSTM unit consists of a cell, an input gate, an output gate, and a forget gate as shown in Figure 5.3. All of which use the sigmoid function that limits the output value between 0 and 1. The gate is considered open if the output stays closer to 1, otherwise it's "closed" if the output is closer to 0. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell:

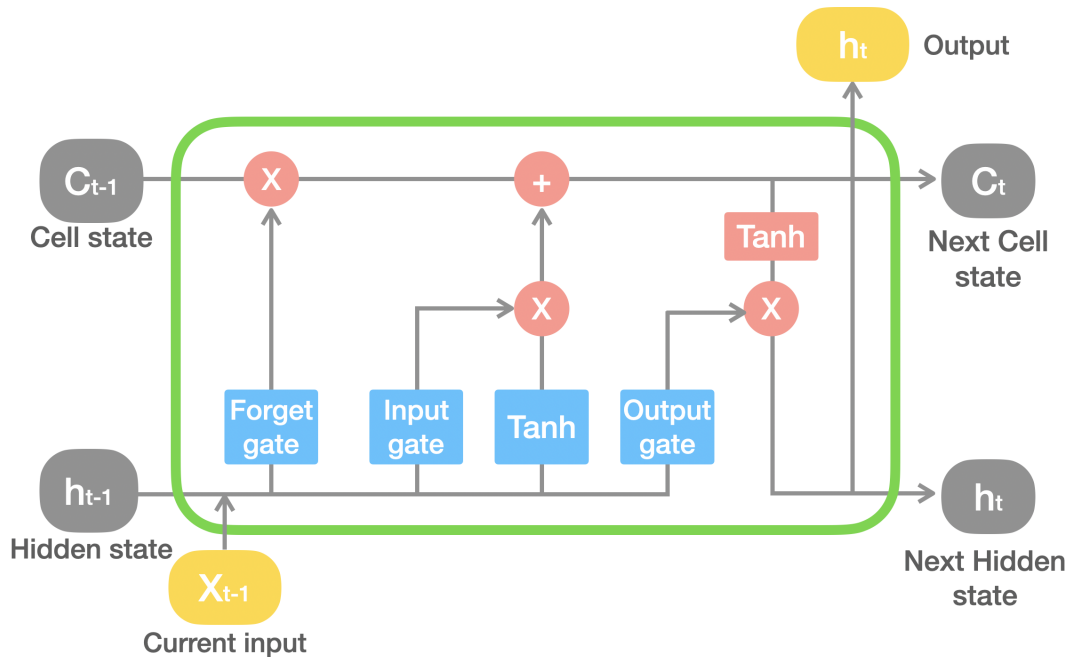


FIGURE 5.3: The structure of LSTM unit. The cell state  $C_{t-1}$  is the memory from last LSTM unit. The hidden state  $h_{t-1}$  is the output of the last LSTM unit.

- The input gate decides the input information that can be sent into the cell state.
- The output gate determines what information should be sent to the hidden state.
- The forget gate is responsible for whether the information should be removed from the previous time step.

The limitation of LSTM is that single LSTM cell could only deal with previous context, not utilize the future context. To solve this problem, Bidirectional Recurrent Neural Networks (BRNN) was introduced by (Schuster and Paliwal, 1997). BRNN combines two separate hidden LSTM layers of two directions, For NLP problem, usually left to right, and right to



left for a given sentence, to the same output. Thus the model could utilize related information from both the previous and future context.

## 5.4 Learning Interactively from Natural Guidance (*LING*)

Our goal is to learn a model that maps the input, which is natural language command and natural language feedback given by non-expert human trainers, to the logical representation of output, that is "Short-Circuit" Geometric Linear Temporal Logic (GLTL) formula. Seq2Seq model is the state of the art solution.

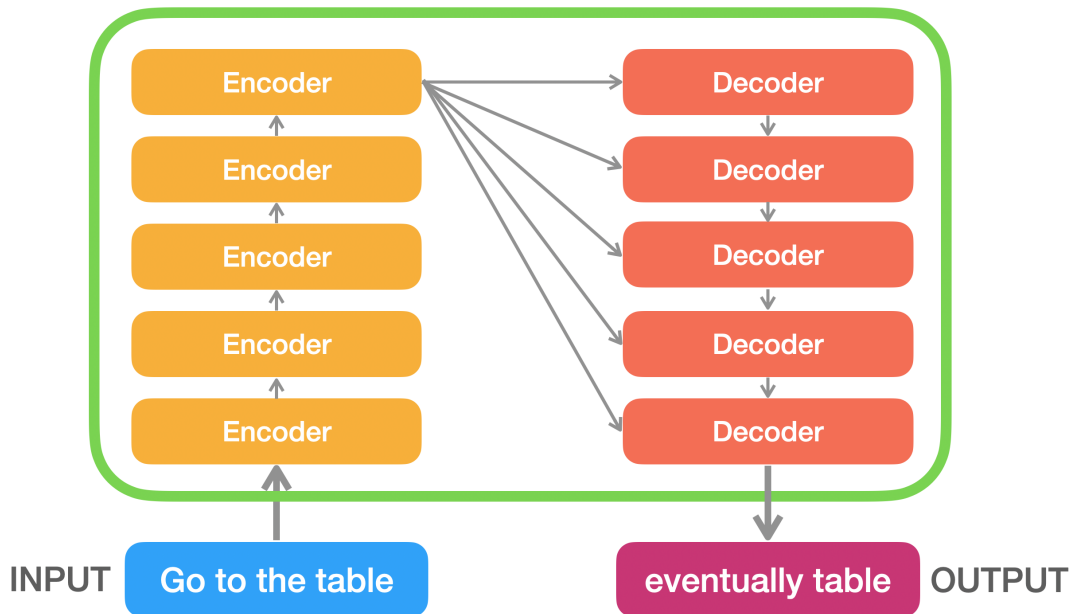


FIGURE 5.4: The high-level structure of encoder, decoder, and the connection between them.

Figure 5.4 demonstrates the high-level structure of encoder, decoder, and the connection between them. The encoder is for input representation which is often a list of vectors. Based on such features matrix, the decoder predicts one symbol at a time until there is an end-of-sentence (EOS) symbol.

The intermediate step, involving attention mechanism, allows the decoder to focus on certain parts of the feature matrix rather than the whole of it. (Dong and Lapata, 2016) use an encoder-decoder model with attention to build a general model for sequence transduction. They encode input utterances into vector representations, and generate their logical forms by conditioning the output sequences to trees on the encoding vectors.

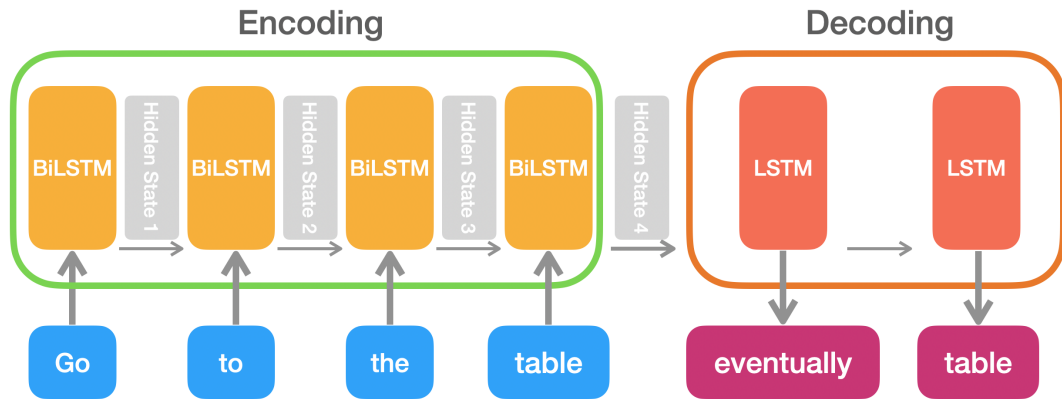


FIGURE 5.5: Detailed visualisation of how encoder-decoder works. Each time step an input unit is taken by the BiLSTM, it updates its hidden state based on its inputs and previous inputs it has seen. The last hidden state of the encoder is the context passed along to the decoder.

A more detailed instruction of how the natural language input "Go to the table" is converted to " $\diamond$ table" is shown as Figure 5.5. The encoder processes each word in the input sequence, it compiles the information it captures into hidden state, which can be represented as a vector. After processing the entire input sequence, the encoder sends the context over to the decoder, which generates the output sequence word by word.

After all the input tokens of the sequence are encoded into vectors, they initialize the hidden vector of the first step of the decoder. To convert a word token into a vector, I make use of word embedding method (Bengio et al., 2003). The semantic meaning and related information of the words, e.g., Paris - France + China = Beijing. The first RNN step takes in the first input vector and initialize the hidden state, and pass on to the second RNN step, which will then process the second input token to update the hidden state. The last hidden state of the encoding component is sent to the decoder.

There is a hidden state that the decoding component uses, that it passes from one time step to the next. Notice that the LSTM of the decoder outputs the top one result greedily by default.

### 5.4.1 Attention Mechanism

The encoder-decoder structure has shown great power. However, when the input sentence is long, a single vector usually can not bring useful information from the encoder to the decoder without causing problems like losing focus or noise. The attention mechanism makes it possible to avoid attempting to learn a single vector for the entire sentence, it focus on a few key input vectors of the sequence based on the attention weights (Bahdanau, Cho, and Bengio, 2014).

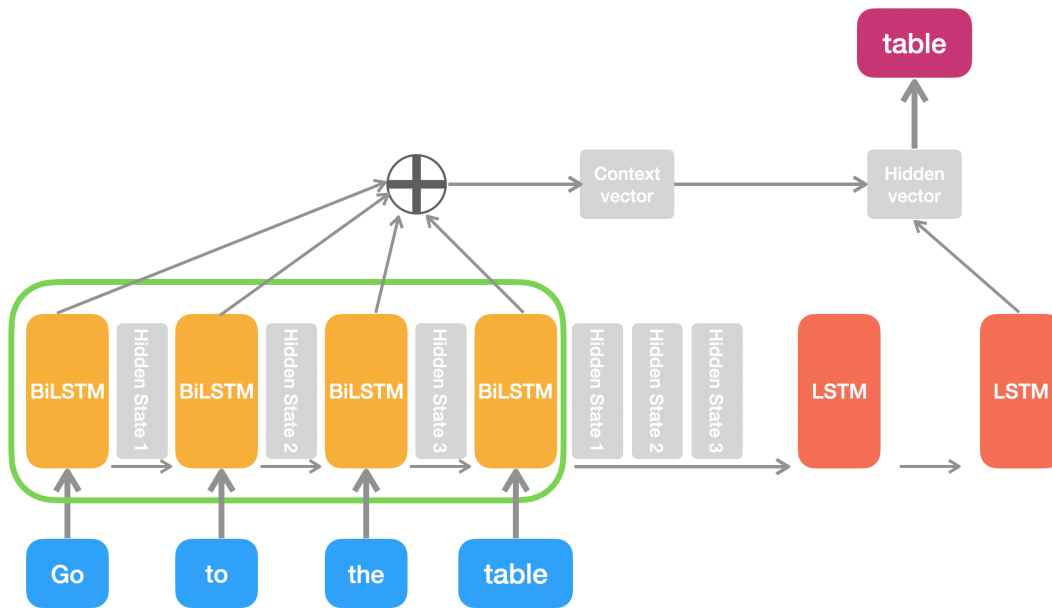


FIGURE 5.6: Structure of the attention mechanism. Attention scores are computed by all the hidden vectors of the encoder and the current hidden vector.

The brief structure of the attention mechanism is given as Figure 5.6. With its help, the decoder will know how much "attention" it needs to pay to the input tokens by the attention weights.

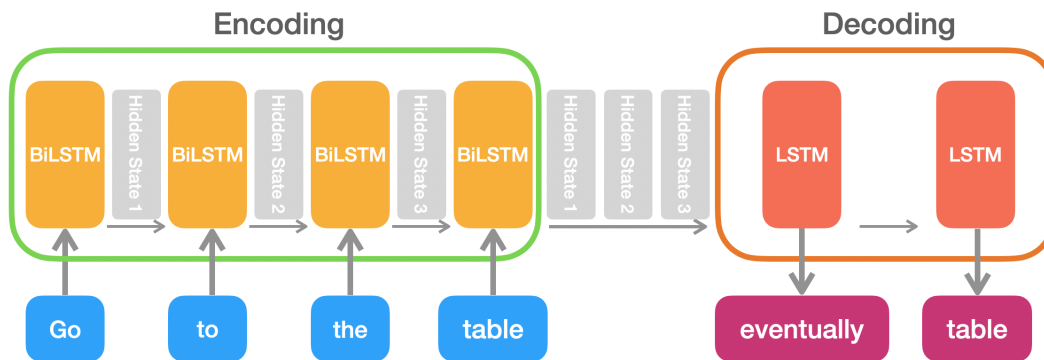


FIGURE 5.7: Encoder-decoder structure with attention. The encoder passes all the hidden states to the decoder. An attention decoder calculates the attention score based on the all the hidden vectors of the encoder and the current hidden vector.

(Bahdanau, Cho, and Bengio, 2014) suggested that:

- All hidden states of the encoder and the decoder help generate the context vector as shown in Figure 5.7, instead of using only the last hidden vector of the encoder otherwise;
- The attention aligns the input and output with an "attention score" which is parameterized by a feed-forward network. The score contributes to showing the most relevant information in the sequence source;

- Every output token is generated based on the context vectors associated with the source position and the previously generated tokens.

Here comes another question: should we pick only the top one greedy output?

### 5.4.2 Beam Search

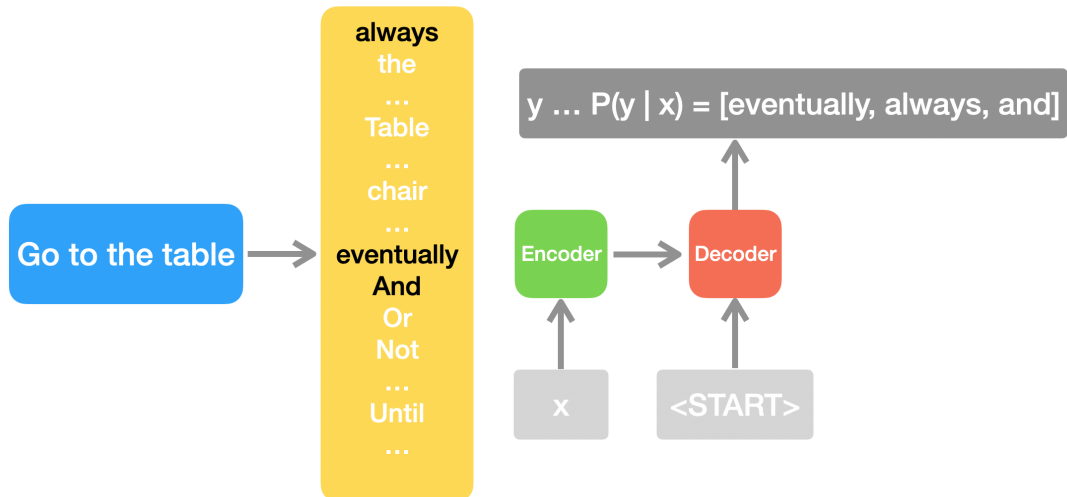


FIGURE 5.8: Given the input sentence, with Beam width set to 3, the model finds the top 3 words with the highest probability. In this example, the outputs are: "eventually", "always", "and".

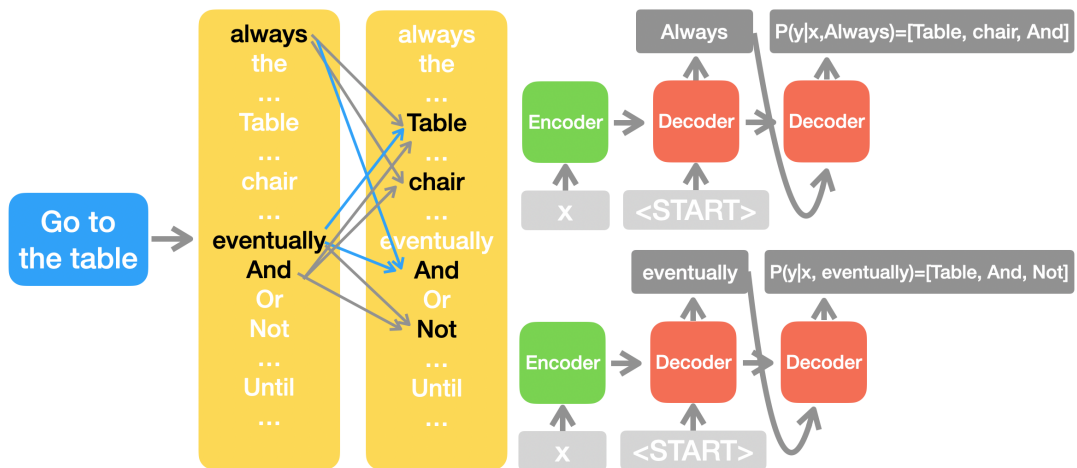


FIGURE 5.9: Given the input sentence, with Beam width set to 3, the model will then find the three best pairs for the first and second words based on conditional probability.

Greedy search selects the best candidate, which is good, but sub-optimal if multiple outputs can be accepted. The beam search algorithm produces multiple candidates alternatively based on conditional probability. The number of the outputs is defined as Beam Width. A detailed introduction is introduced in Figure 5.8 and Figure 5.9:

In this case, let's use one natural language command from the experiment, "Go to the table" as an input example. The target GLTL output of which is  $\diamond$ table. The beam width is set to 3.

1. The encoder converts the input sequence to hidden states, and passes that information to the decoder, which then applies softmax function to the whole vocabulary;
2. The three words with the highest probability are selected since the beam width is 3. In Figure 5.8, the top three candidates are "eventually", "always", "and". If the beam width is set to 1, then it's the same as greedy search;
3. As shown in Figure 5.9, the algorithm will then find the three best pairs for the first and second words based on conditional probability. The top three words selected at previous step, "eventually", "always", "and" are taken as input, each of which will generate a small list pair of the first and second words. For example, we get "always Table", "always chair", and "always And" from the first word "always", and "eventually Table", "eventually chair", and "eventually Not" by the first word "eventually". After the softmax function, the top three pairs of all candidates will be produced based on conditional probability: "always And", "eventually Table", and "eventually And";
4. Sending "always And", "eventually Table", and "eventually And" as input to the next round to find the best three output candidates for the first, second, and third word in the same way;
5. Repeat the process until the signal of "End Of Sentence" (EOS) appears.

With the power of beam search algorithm, now it's possible to generate multiple best candidates from the input sequence, which will contribute a lot to the GLTL-based algorithm introduced in Chapter 4.

### 5.4.3 Algorithms

With the advantages of the GLTL-based framework, Seq2Seq model, and the ideas from COACH, I introduce the *Learning Interactively from Natural Guidance (LING)* method shown as algorithm 7. *LING* takes in the environment information (atomic propositions), voice command from the human trainer as input, and learns from reinforcement, which is generated by the voice feedback provided by human trainers during training.

I choose a Seq2Seq model with Bahdanau attention mechanism (Bahdanau, Cho, and Bengio, 2014), taking Bidirectional Long Short-Term Memory (BiLSTM) (Schuster and Paliwal, 1997) as encoder (Cho et al., 2014a), LSTM as decoder, and apply Beam Search Mechanism to generate the top  $k$  outputs.

The main reasons that I use this set up are:

- The training speed is acceptable comparing to other approaches like transformer-based models (Vaswani et al., 2017). The parameter space of the chosen model is much smaller, which makes it significantly faster.
- The inference speed is also acceptable to humans.
- The performance is acceptable. Other approaches like transformer-based models may have better accuracy, but that is not prioritized in this problem.

The model considers the input and output both as a sequence of information. The encoding component is a stack of encoders. The decoding component is a stack of decoders of the same number.

Algorithm 3 represents the case when the trainer only gives evaluative feedback instead of natural language feedback.

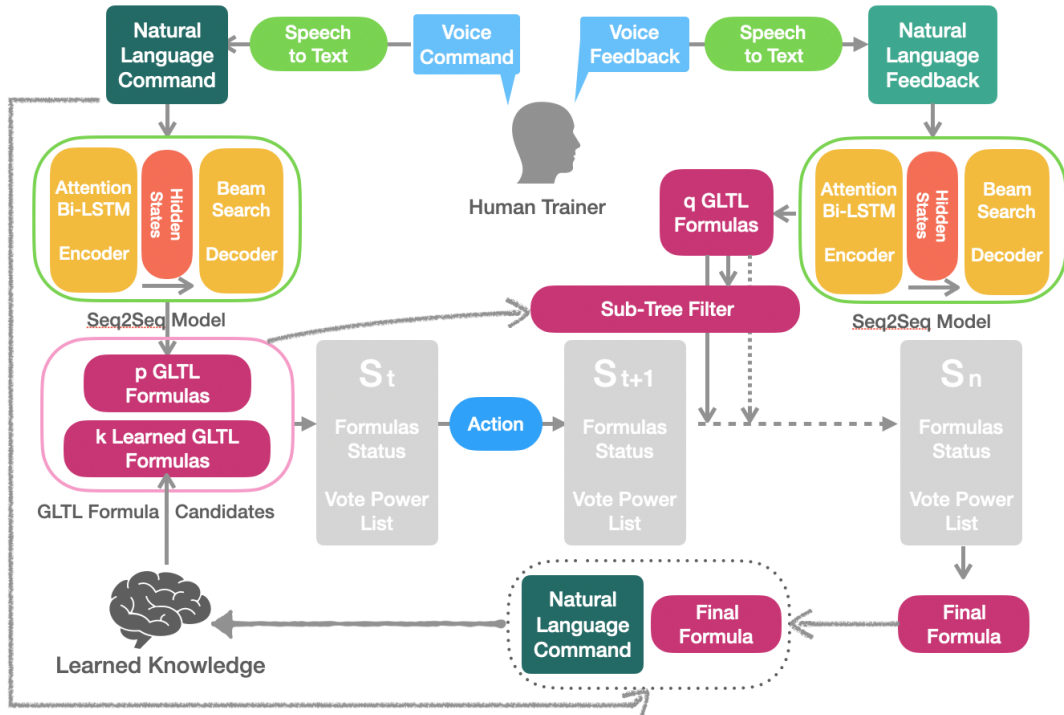


FIGURE 5.10: Learning Interactively from Natural Language Model (*LING*).

1. Before the start of the training process, both the trainer and the learner know about the environment, which include the names and positions of the atomic propositions, and also the action space of the agent in the environment;
2. During the very first time of training, the trainer needs to give an example of his preferred positive reward, like saying "good job!" in English, and another example of his preferred negative reward, for example, saying "no!" in English to the agent. The voice commands will be converted to natural language text commands through Speech-to-Text model (Graves and Jaitly, 2014). In this work, I use Google's API (Schalkwyk et al., 2010). Other languages like Chinese, Spanish are also acceptable as long as the speech-to-text converter can support them;
3. When the trainer wants to teach the agent a mission, he can tell the mission to the agent, the Seq2Seq model will then generate the top  $p$  candidate GLTL formulas;
4. If the trainer considers the mission to be too complicated or challenging to the agent, he can decompose the mission to a set of simpler sub-tasks, tell the agent the first sub-task and train the agent to learn it, then the second sub-task. The trainer can keep teaching the sub-tasks until the agent learns and finishes the entire mission;
5. The *Learned Knowledge* of the agent, which is a list of pairs of natural language command and the learned corresponding GLTL formula, keeps the record of all the tasks that have ever been learned by the agent. Thus the agent has a lifelong memory that could be transferred to future tasks. The default formulas are determined by the environment and the template 4.1;
6. When a command is given by the trainer, the *Learned Knowledge* will generate  $k$  candidate formulas, adding up to the final candidate pool of  $p + k$  GLTL formulas. All

the valid sub-formulas of the candidate formulas of the  $p + k$  pool build up a *Sub-Tree Filter* to help the agent understand the natural language feedback during the training process;

7. The agent follows the GLTL-based algorithm, letting the  $p + k$  candidate formulas vote for an action to take;
8. At anytime during training, the human trainer can give feedback by speaking to the agent, the natural language feedback that is converted by the Speech-to-Text model will be sent to the Seq2Seq model, which generates top  $q$  candidate GLTL formulas accordingly;
9. The shared formulas between the  $q$  feedback formulas and the *Sub-Tree Filter* will then be used to update the dynamic *Vote Power List* as shown in Figure 5.10 and explained in Algorithm 7;
10. When the trainer is satisfied with the behavior of the agent, or considers the mission to be finished, he can stop the agent with a finishing signal;
11. After the training process is finished, the agent forms a new pair of the original natural language command and the final learned GLTL formula, adding the pair to the *Learned Knowledge* and update the Seq2Seq model;
12. Now the agent has learned a new command from the human trainer, and it will perform better when a similar command appears in the future, since the Seq2Seq model and the *Learned Knowledge* are both updated. In a word, it becomes a bit smarter.

The *Learned Knowledge* enables the agent to learn in a lifelong way:

- If allowed by the human trainer, his own agent can share the *Learned Knowledge* with other agents, so that every training effort of every trainer could benefit the entire community. The agent becomes smarter overtime, that most tasks may have been learned in other scenarios;
- The *Learned Knowledge* enables the agent to adapt to the behavioral model of the human trainer. It could understand the expected performance of a personal-style command.

## 5.5 Experiment

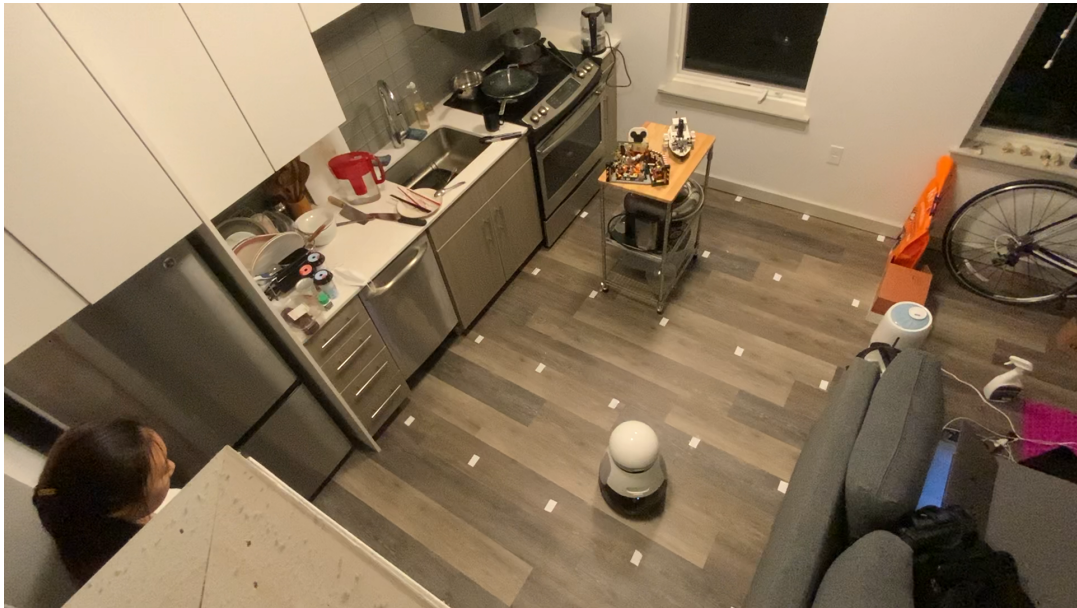


FIGURE 5.11: The experiment space for non-expert user study. The corners of the virtual grids are marked with small white paper cards on the floor.

An user study with non-expert human trainer was conducted to study the performance of *LING*. Due to COVID-19, the campus was closed, which stopped the participants from coming to the lab to finish the experiment. Therefore, I built an experiment space in an apartment living room as shown in Figure 5.11.



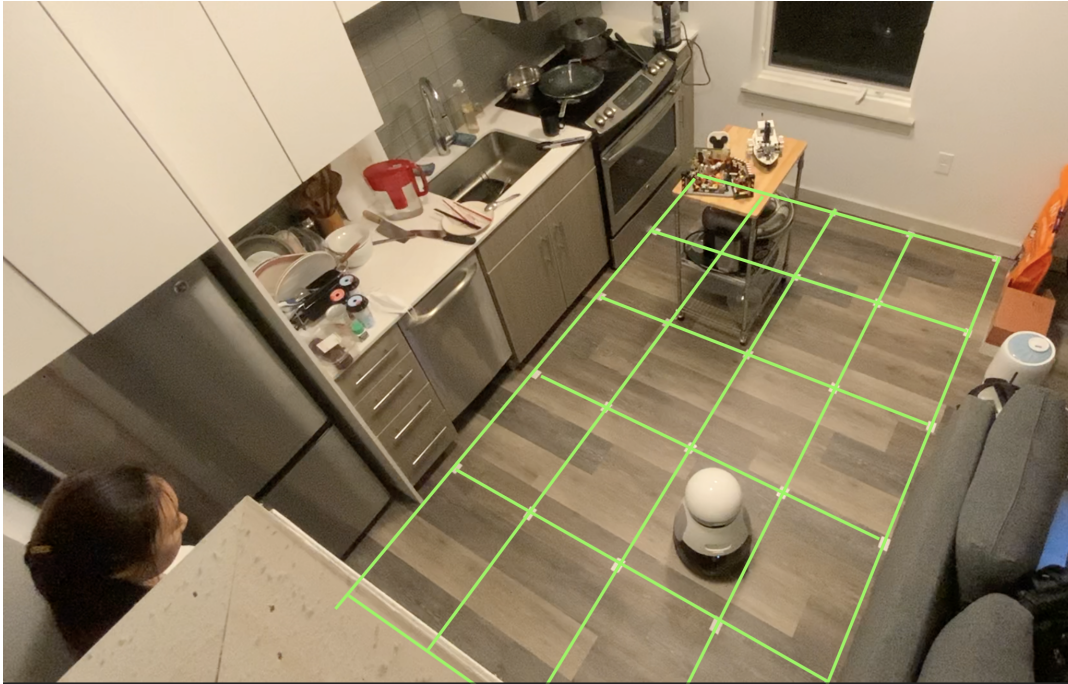


FIGURE 5.12: The experiment space for non-expert user study. The 4 by 5 grid world is marked in green virtual lines. There are five atomic propositions in the environment that are known by both the human trainer and the agent: The charger, the fridge, the oven, the window, and the "table" which is actually a kitchen island.

The floor was set up to a 4 by 5 grid world. Each grid was 44 cm by 61 cm, with four corners marked by small white paper cards on the floor. The virtual grids are marked using green lines in Figure 5.12. Since it's a real studio, it's actually very similar to the scene where an intelligent robot is trained by its owner to finish daily tasks. There are a few common objects in the space, five of which are considered as atomic propositions:

- The charger, which is at the bottom-right corner next to the couch, hidden in Figure 5.12 next to the wall. The location of the charger is  $x = 3, y = 0$ ;
- The oven, which is at the left side of the space. The location of the oven is  $x = 0, y \in [3, 4]$ ;
- The window, which is at the top side of the space. The location of the window is  $x \in [1, 2], y = 4$ ;
- The fridge, which is at the bottom-left of corner of the space. The location of which is  $x = 0, y = 0$ ;
- The "table", which is actually a kitchen island in the space. The location is  $x = 1, y = 3$ .

All other objects are considered as background by both the agent and the trainer.



FIGURE 5.13: The photo of Kuri, a robot developed by Mayfield Robotics.

I deployed *LING* algorithm to a robot called Kuri ([“Kuri Robot Website”](#)). Kuri 5.13, according to the company Mayfield that developed it, is "an intelligent robot for the home". It is about half a meter tall, weights over 6 kilograms, and is equipped with a camera sensor to collect visual information from its front, a microphone array to collect voice, speakers, and touch sensors. Kuri uses a mixed solution of "laser-based sensor array" to detect obstacles, localize itself, memorize the environment as a map, and navigate.

The robot uses a vacuum-like drive system to move on the floor. Three types of actions are supported:

- move forward.
- move backward.
- rotate.

For the implementation of *LING*, the action space was converted to

- move up.
- move right.
- move down.
- move left.
- stay.

At each step, the robot can only move to the grid cell next to its current position in the same row or column, or choose to stay. After each action, it will wait for the computation of

the dynamic *Vote Power List* to be done so that it knows the next action to take. The trainer can say anything during training, including saying nothing. When the trainer thinks that the agent has learned or has successfully finished the mission, she can say "Finish!" to end the training round.

Kuri was controlled through the Robot Operating System (ROS). Based on the functionalities of Kuri and the design of the algorithm *LING*, I used the mapping function of Kuri to locate the agent, and did not make use of other sensors. The *Learned Knowledge* space of the agent was set to empty before the first task. The command that it has learned during each task will be added to the *Learned Knowledge* space. The *Learned Knowledge* will be kept and carried on to all future tasks.

Before the start of the experiment, the participant was given a research consent, and was asked to read it carefully before signing it. The experiment environment was sanitized to protect people from COVID-19. Both the researcher and the participant were well prepared and protected following the regulations provided by Brown University.

A video tutorial was displayed for the participant, which introduced the goal of the experiment, and showed an example of the entire process of training a complex mission through decomposition. The participant was then asked to provide an example of her preferred way of giving positive feedback orally, which is "Good job!", and another example of negative feedback, "No!" to the agent after watching the tutorial.

The missions that the participant need to teach the agent were:

- Train the robot to go to the window then go to the fridge.  $\diamond(window \wedge \diamond fridge)$ ;
- Go to the window without touching the oven.  $\neg oven \mathcal{U} window$ ;
- Go between the window and the fridge.  $\square \diamond (window \wedge \diamond fridge)$ ;
- Never touch the window until you arrive at the oven.  $\neg window \mathcal{U} oven$ .

After the participant felt confident about what to do, she was given the first mission. The participant was asked to decide the task that she would like to train at each round, and she made it:

1. Go to the window.
2. Go to the fridge.
3. Go to the window then go to the fridge.

The feedbacks that she gave to the agent during the first mission include: "Good job!", "Keep going!", "This is wrong", "No!", "Go back!", "Yes!", "No, it's the wrong way.", "Great!" which are similar to the natural language feedback from the tutorial video. Commands like "Turn right!", "Turn to your right!", "One more step!", "Turn left, two more steps!" were also given to the robot.

Considering the fact "Turn left" contains information that relies on not only the policy, but also the current status of the agent, it brings external information to the agent. When the agent has learned more tasks, such natural language feedback may become useful to help the agent converge to the target policy faster.

The trainer successfully trained the agent to learn all of the four missions within thirty-five minutes. There's a clear trend that the training time for each task became less as the agent learned more things. On the other hand, the trainer was very passionate and gave feedback very frequently at the beginning, and became more quiet giving less feedback as the experiment went on. The agent successfully learned from both the rich type and sparse type of feedback with the help of the two eligibility traces inherited from COACH in Chapter 3.

During training, sometimes the trainer gave wrong feedback without noticing what happened. During the third mission, however, she realized that she had asked the robot to do wrong things, and gave very long and complicated feedback to the agent trying to fix it. The wrong and long feedback slowed the upgrade of the *Vote Power List* a little bit. However, since the Seq2Seq model did not convert them into correct GLTL formulas, only very limited noises were added.

After all four missions were finished, the trainer was asked to fill in a post-experiment survey.

**Algorithm 3** LING with Evaluative Feedback

---

```

function LEARNTASK( $X$ , SHARPENING  $\alpha$ , TRACE_RIGHT  $\lambda_T$ , TRACE_WRONG  $\lambda_F$ ,
DELAY  $D$ , FINISH DELAY  $FD$ )
   $restarts \leftarrow 0$ ,  $L \leftarrow X$ 
  for  $\phi \in X$  do
     $r_\phi \leftarrow 0$ , initialize reward vector
     $v_\phi \leftarrow 1/|X|$ , initialize vote vector
     $finish_\phi \leftarrow -1$ , number of steps after goal achieved. If not achieved, get -1
     $e_{T\phi 0} \leftarrow 0$ , trace vector for positive rewards, time added
     $e_{F\phi 0} \leftarrow 0$ , trace vector for negative rewards, time added
  end for
   $t \leftarrow 0$ , choose starting state  $S_0$ 
  while task has not ended do observe current state  $S_{t-1}$ 
    for all  $a \in A$  do  $c_a = \text{sum}(v_\phi)$  for  $\phi$  s.t.  $a$  is the optimal policy in  $X_\phi$ 
    end for
    execute  $a_t = \text{argmin}_a |c_a - 0.5|$ , state becomes  $S_t$ 
     $finish \leftarrow \text{UpdateFinishState}(finish, S_t, X)$ 
    for all  $a \in A$  do
      if  $a_t$  is the optimal policy in  $X_\phi$  then
         $e_{T\phi t} \leftarrow \lambda_T * e_{T\phi(t-1)} + 1$ 
         $e_{F\phi t} \leftarrow \lambda_F * e_{F\phi(t-1)} + 1$ 
      else
         $e_{T\phi t} \leftarrow \lambda_T * e_{T\phi(t-1)} - 1$ 
         $e_{F\phi t} \leftarrow \lambda_F * e_{F\phi(t-1)} - 1$ 
      end if
    end for
    if trainer attempts to end the task then  $r \leftarrow \text{RewardFinish}(r, finish, X, d, fd)$ 
    if  $|L| = 1$  or  $restarts \geq 10$  then break
    else
       $restarts \leftarrow restarts + 1$ 
      for  $\phi \in X$  do
         $finish_\phi \leftarrow -1$ , number of steps after goal achieved. If not, get -1
         $e_{T\phi 0} \leftarrow 0$ , trace vector for positive rewards
         $e_{F\phi 0} \leftarrow 0$ , trace vector for negative rewards
      end for
       $t \leftarrow 0$ , choose starting state  $S_0$  s.t. most distinctive in  $|L|$ 
    end if
    else if Trainer gives feedback  $f_t$  then  $r \leftarrow \text{PenalizeFinish}(r, finish, X, d, fd)$ 
    if  $f_t > 0$  then
      for all  $\phi \in X$  do  $r_\phi \leftarrow r_\phi + f_t * e_{T\phi(t-d)}$ 
      end for
    end if
    else if  $f_t < 0$  then
      for all  $\phi \in X$  do  $r_\phi \leftarrow r_\phi + f_t * e_{F\phi(t-d)}$ 
      end for
    end if
     $v = \text{softmax}(\alpha * r)$ , update vote
     $L = \{\phi \in X | r_\phi = \min_{j \in X} r_j\}$ , policy with maximum cumulative reward
  end while
  return  $L$  (at most 2))
end function

```

---

---

**Algorithm 4** LING Supplemental Function Update Finish State

---

```

function UPDATEFINISHSTATE(finish,  $S_t$ , X)
  for all  $\phi \in X$  do
    if  $finish_\phi > -1$  then
       $f_\phi \leftarrow finish_\phi + 1$ 
    else
      if policy  $\phi$  ends at  $S_t$  then
         $finish_\phi \leftarrow 0$ 
      end if
    end if
  end for
  return r
end function

```

---



---

**Algorithm 5** LING Supplemental Function Reward Finish State

---

```

function REWARDFINISHSTATE( $R$ , finish, POLICIES X, DELAY D, FINISH_DELAY FD)
  for  $\phi \in X$  do
    if  $d \leq finish_\phi \leq d + fd$  then
       $r_\phi \leftarrow r_\phi + 1$ 
       $finish_\phi \leftarrow -1$ 
    end if
  end for
  return r
end function

```

---



---

**Algorithm 6** LING Supplemental Function Penalize Finish State

---

```

function PENALIZEFINISHSTATE( $R$ , finish, POLICIES X, DELAY D, FINISH_DELAY FD)
  for  $\phi \in X$  do
    if  $d \leq finish_\phi \leq d + fd$  then
       $r_\phi \leftarrow r_\phi - 1$ 
       $finish_\phi \leftarrow +1$ 
    end if
  end for
  return r
end function

```

---

**Algorithm 7** LING with Natural Language Feedback

**function** LEARNTASK( $X$ , SHARPENING  $\alpha$ , TRACE\_RIGHT  $\lambda_T$ , TRACE\_WRONG  $\lambda_F$ ,  
DELAY  $D$ , FINISH DELAY  $FD$ , DISCOUNT FACTOR  $\theta$ )

$restarts \leftarrow 0$ ,  $L \leftarrow X$

**for**  $\phi \in X$  **do**

$r_\phi \leftarrow 0$ , initialize reward vector

$v_\phi \leftarrow 1/|X|$ , initialize vote vector

$finish_\phi \leftarrow -1$ , number of steps after goal achieved. If not achieved, get -1

$e_{T\phi 0} \leftarrow 0$ , trace vector for positive rewards, time added

$e_{F\phi 0} \leftarrow 0$ , trace vector for negative rewards, time added

$F_\phi \leftarrow 0$ , initialize natural language feedback vector

**end for**

$t \leftarrow 0$ , choose starting state  $S_0$

**while** task has not ended **do** observe current state  $S_{t-1}$

**for all**  $a \in A$  **do**  $c_a = \text{sum}(v_\phi)$  for  $\phi$  s.t.  $a$  is the optimal policy in  $X_\phi$

**end for**

execute  $a_t = \text{argmin}_a |c_a - 0.5|$ , state becomes  $S_t$

$finish \leftarrow \text{UpdateFinishState}(finish, S_t, X)$

**for all**  $a \in A$  **do**

**if**  $a_t$  is the optimal policy in  $X_\phi$  **then**

$e_{T\phi t} \leftarrow \lambda_T * e_{T\phi(t-1)} + 1$

$e_{F\phi t} \leftarrow \lambda_F * e_{F\phi(t-1)} + 1$

**else**

$e_{T\phi t} \leftarrow \lambda_T * e_{T\phi(t-1)} - 1$

$e_{F\phi t} \leftarrow \lambda_F * e_{F\phi(t-1)} - 1$

**end if**

**end for**

**if** trainer attempts to end the task **then**  $r \leftarrow \text{RewardFinish}(r, finish, X, d, fd)$

**if**  $|L| = 1$  or  $restarts \geq 10$  **then break**

**else**  $restarts \leftarrow restarts + 1$

**for**  $\phi \in X$  **do**

$finish_\phi \leftarrow -1$ , number of steps after goal achieved. If not, get -1

$e_{T\phi 0} \leftarrow 0$ , trace vector for positive rewards

$e_{F\phi 0} \leftarrow 0$ , trace vector for negative rewards

**end for**

$t \leftarrow 0$ , choose starting state  $S_0$  s.t. most distinctive in  $|L|$

**end if**

**else if** Trainer gives feedback  $f_t$  **then**  $r \leftarrow \text{PenalizeFinish}(r, finish, X, d, fd)$

**if**  $f_t > 0$  **then**

**for all**  $\phi \in X$  **do**  $r_\phi \leftarrow r_\phi + f_t * e_{T\phi(t-d)}$

**end for**

**end if**

**else if**  $f_t < 0$  **then**

**for all**  $\phi \in X$  **do**  $r_\phi \leftarrow r_\phi + f_t * e_{F\phi(t-d)}$

**end for**

**end if**

**for all**  $\phi \in (L \cap F)$  **do**  $r_\phi = (1 + \theta) * r_\phi$

**end for**

**for all**  $\phi \in (F - (L \cap F))$  **do**  $r_\phi = \theta$ ,  $L = L \cup F$

**end for**

$v = \text{softmax}(\alpha * r)$ , update vote

$L = \{\phi \in X | r_\phi = \min_{j \in X} r_j\}$ , policy with maximum cumulative reward

**end while**

return  $L$  (at most 2))

**end function**





## Chapter 6

# Conclusions and Future Directions

The core of this thesis are:

- Humans.  
Specifically, non-expert people who may not have any background of math, computer science, programming, robotics, or machine learning.
- Tasks.  
From simple to complex, tasks of arbitrary complexity.
- Learning from interaction.  
The agent learns from interacting with the environment and receives feedback from human trainer directly.
- Natural guidance.  
A combination of evaluative feedback, natural language command, and natural language feedback given by humans.

### 6.1 Humans

Humans are the problem. We make mistakes all the time.

In Chapter 3, I demonstrate that human feedback is policy-dependent. Unlike sparse goal feedback and stationary action feedback, the feedback that non-expert human trainer gives to the agent possesses properties of:

- Diminishing returns.  
As the learner's performance improves, trainer feedback decreased—an alternative explanation is simply trainer fatigue.
- Differential feedback.  
The strength of feedback varies with improvement and deterioration.
- Policy shaping.  
It reinforces sub-optimal actions, then punishes them and raises the bar.

The *Advantage Function* is introduced to simulate the properties of policy-dependent human feedback. The *Advantage Function* describes how much better or worse an action selection is compared to the agent's performance under policy  $\pi$ .

Experiments in Chapter 4 showed that human trainers make mistakes quite often. About one third of all the feedback were wrong. The algorithms proposed in this thesis are designed to learn from such noisy human feedback.

## 6.2 Tasks

Task representation is one of the most important parts of reinforcement learning. Unlike traditional RL methods that try to maximize the reward function, Chapter 2 introduces "Short-Circuit" Geometric Linear Temporal Logic (GLTL) as a way of representing tasks of arbitrary complexity.

GLTL can represent temporal information. A GLTL formula, which can be shown as a tree structure, can be decomposed to smaller pieces. Simpler tasks can build up to complex tasks in an efficient and effective way.

## 6.3 Learning from interaction

Three algorithms are proposed in the thesis:

- CONvergent Actor-Critic by Humans (COACH)
- "Short-Circuit" GLTL-based algorithm
- Learning Interactively from Natural Guidance (LING)

### 6.3.1 COACH

In Chapter 3, COACH is introduced to learn from policy-dependent evaluative human feedback. COACH is based on the insight that the *Advantage Function* (a value roughly corresponding to how much better or worse an action is compared to the current policy) provides a better model of human feedback, capturing human-feedback properties like diminishing returns, rewarding improvement, and giving 0-valued feedback a semantic meaning that combats forgetting.

Experiments in simulation environment compared the performance of COACH, Q Learning, and TAMER, where each algorithm is tested with three different types of feedback: sparse goal reward, stationary action reward, and improvement reward which is close to evaluative human feedback. The results suggest that COACH with eligibility traces learns robustly with all three kinds of feedback, while Q Learning and TAMER could only work with limited feedback type.

A qualitative experiment with a physical robot, TurtleBot, which runs Real-time COACH, a special version of COACH, is conducted. The experiment suggests that COACH can scale to a complex domain involving multiple challenges. Coach successfully learned all of the five tasks while TAMER fail to learn the compositional behaviors:

1. push-pull.  
To teach the agent to navigate to the ball when it is far, and back away from it when it's near.
2. hide.  
To teach the agent to go away from the ball and turn away when it moves far.
3. ball following.  
To teach the agent to come to the ball.
4. alternate.  
To teach the agent to go back and forth between the ball and the cylinder.
5. cylinder navigation.  
To teach the agent to navigate to the cylinder with the help of the ball as a lure.

### 6.3.2 GLTL-based algorithm

Chapter 4 presents a GLTL-based algorithm to learn tasks of arbitrary complexity through decomposition. The results of simulation experiments show that the feedback it takes for the GLTL-based algorithm to finish four simple tasks is much fewer than the other two algorithms.

In another simulation experiment, GLTL-based algorithm can handle all of the five complex tasks well while both TAMER and COACH fail in learning:

1. Go to the table and then go to the charger and stay there:  $\diamond(\text{table} \wedge \diamond\Box\text{charger})$ .
2. Go to your charger without colliding with either the chair or the table:  $\neg(\text{chair} \vee \text{table}) \mathcal{U} \text{charger}$ .
3. Go to the table and then go to the fridge:  $\diamond(\text{table} \wedge \diamond\text{fridge})$ .
4. Go to the charger and then go to the chair without running into the table along the way:  $(\neg\text{table}) \mathcal{U}(\text{charger} \wedge \diamond\text{chair})$ .
5. Go back and forth patrolling between the table and the fridge:  $\Box\diamond(\text{table} \wedge \diamond\text{fridge})$ .

### 6.3.3 Similarity score

To measure the acceptance level of the learned formula, I introduce *Similarity Score* as evaluation metric. The *Similarity Score* of two formulas is calculated by counting the percentage of positive feedback of all trajectories of one formula from an ideal trainer trying to teach the other formula. It can be thought of as telling us how happy someone would be when they were wanting to train a formula and the robot exhibits behavior from the other formula.

If the *Similarity Score* of an output learned formula and its original target formula is close to 1, it means that the actual behavior of the agent is close to the desired behavior, therefore the acceptance level can be considered high.

A series of user studies with real participants suggest that:

1. Non-expert people can decompose complex tasks to simpler sub-tasks effectively.
2. Complex missions can be taught by non-expert human trainers through decomposing them to simpler sub-tasks, and training the GLTL-based agent sub-tasks to build up the missions via evaluative feedback.

### 6.3.4 LING

In Chapter 5, a deep sequence-to-sequence (Seq2Seq) approach is used to interpret natural language command and feedback. The Seq2Seq algorithm is applied to convert natural language to formal language, which is GLTL formula. The key factors in deciding the structure of the Seq2Seq model are:

- Training cost.  
The model needs to learn fast over small training set, so that it's possible for human teacher to train a new model from scratch.
- Training speed.  
The model needs to be trained fast so that the human trainer does not need to wait for the agent to update its model.

As a result, a Seq2Seq model with attention mechanism is selected, taking Bidirectional Long Short-Term Memory (Bi-LSTM) as encoder, LSTM as decoder, applying beam search to generate multiple top outputs.

Based on the idea of COACH, GLTL, and Seq2Seq model, I introduce an approach of *Learning Interactively from Natural Guidance (LING)* that enables the agent to learn tasks of arbitrary complexity through decomposition and a combination of evaluative feedback and natural language feedback.

*LING* can learn from both evaluative numerical feedback and natural language feedback. It could parse the natural language command as a group of possible GLTL formula candidates, and combine the group with the *Learned Knowledge* to form the potential candidates list. All the sub-formulas of the potential candidates list produce an extended list, which will be used as a filter. The agent will then take actions generated by the dynamic voting function of the list. Each natural language feedback that the human trainer gives along the way are translated to several GLTL formulas picked by the filter, and then get assigned to the list through multiple eligibility traces.

At the end of the task, the learned formula and the original natural language command will be applied to update the Seq2Seq model and the *Learned Knowledge* base.

A qualitative user study suggests that non-expert human trainer can teach physical robot running *LING* to complex tasks in a real living room by giving natural language commands and natural language feedback.

## 6.4 Natural Guidance

The natural ways that human trainers use to guide the robots are considered to contain three properties in this thesis:

- Evaluative feedback.  
Positive feedback like a "treat" to a puppy, and negative feedback like a beam alarm.
- Natural language command.  
The human trainer could speak to the robot asking it to finish a goal in natural language.
- Natural language feedback.  
The human trainer could speak to the robot to adjust its behavior during the training process.

Apparently such natural guidance can easily be noisy, containing a lot of wrong feedback and unrelated information. However, it's human nature. In general, making the experience of teaching a robot as easy and straightforward as that of teaching a dog requires the ML approach to be able to learn from such natural guidance.

## 6.5 Future Directions

The methods proposed in this thesis have great potentials yet to be explored.

First of all, the infrastructure of *LING* does not make use of pre-trained embedding techniques. The language models pre-trained on huge data sets, e.g., GPT-3, have shown exciting power in many areas. Combining such models to the pipeline could be promising in the way that the agent can expand its knowledge space bigger, richer, and faster.

Secondly, *LING* is a general model that supports multiple languages. If individual agent of *LING* can share their knowledge base through cloud services, then the level of intelligence of the agents will improve very quickly. Every natural language command that has ever been taught by any human trainer speaking whatever language will be learned by all agents.

Furthermore, there are a lot of robotics functions remaining to be discovered and added to the current system. Perception, for example, will enable the robot to watch the environment through visual sensors. The spatial information of the new objects detected and recognized by vision methods can be appended to the atomic proposition list, which will make the user experience much better.

Moreover, *LING* is designed based on the assumption that both the human trainer and the learner, which is the robot agent, know about the environment, which means there's clear limitation of what a robot can do in a given domain. Expanding the limited space, for example, teaching the robot to do tasks of arbitrary complexity in an open environment where the environmental information is partially known by the agent is worth exploring.

Last but not least, *LING* can be applied to not only human-robot interaction, but also other scenarios. As long as the problem can be formed in a way that the agent interacts with a consistent representation of human behavior, *LING* and its customized version might fit in.

Automatic grading the handwritten solutions to K-12 mathematical or physical questions, for example, can be considered as the agent interacts with each step of the image segments of the student's handwritten solution, trying to understand the student's behavior, and generating judgement results of whether each step of the answer is correct. Solving such a problem and producing a solution to the general public will create great social impact, e.g., protecting education fairness.

*“For an agent that can interactively learn from human guidance, the history is meaningful, and the future is learnable.”*



# Bibliography

- Abbeel, Pieter and Andrew Y Ng (2004). “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the Twenty-First International Conference on Machine Learning*.
- (2005). “Exploration and apprenticeship learning in reinforcement learning”. In: *Proceedings of the 22nd international conference on Machine learning*, pp. 1–8.
- Akrour, Riad, Marc Schoenauer, and Michèle Sebag (2011). “Preference-based policy learning”. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 12–27.
- Almagor, Shaull, Udi Boker, and Orna Kupferman (2014). “Discounting in LTL”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 424–439.
- Amershi, Saleema et al. (2014). “Power to the people: The role of humans in interactive machine learning”. In: *AI Magazine* 35.4, pp. 105–120.
- Amin, Kareem, Nan Jiang, and Satinder Singh (2017). “Repeated Inverse Reinforcement Learning”. arXiv preprint arXiv:1705.05427.
- Amsters, Robin and Peter Slaets (2019). “Turtlebot 3 as a robotics education platform”. In: *International Conference on Robotics in Education (RiE)*. Springer, pp. 170–181.
- Artzi, Yoav and Luke Zettlemoyer (2011). “Bootstrapping semantic parsers from conversations”. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 421–432.
- Ayala, Angel, Claudio Henríquez, and Francisco Cruz (2019). “Reinforcement learning using continuous states and interactive feedback”. In: *Proceedings of the 2nd International Conference on Applications of Intelligent Systems*, pp. 1–5.
- Babes, Monica et al. (2011). “Apprenticeship Learning About Multiple Intentions”. In: *Proceedings of the International Conference on Machine Learning*, pp. 897–904.
- Bacchus, Fahiem, Craig Boutilier, and Adam Grove (1996). “Rewarding Behaviors”. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. AAAI Press/The MIT Press, pp. 1160–1167.
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473*.
- Baier, Christel and Joost-Pieter Katoen (2008). *Principles of Model Checking*. MIT Press.
- Baird, Leemon (1995). “Residual algorithms: Reinforcement learning with function approximation”. In: *Proceedings of the twelfth international conference on machine learning*, pp. 30–37.
- Barto, A.G., R.S. Sutton, and C.W. Anderson (1983a). “Neuronlike adaptive elements that can solve difficult learning control problems”. In: *Systems, Man and Cybernetics, IEEE Transactions on SMC-13.5*, pp. 834–846.
- Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson (1983b). “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems”. In: *IEEE Transactions on Systems, Man, and Cybernetics SMC-13.5*, pp. 834–846.
- Bellman, Richard (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.
- Bender, Emily M and Alexander Koller (2020). “Climbing towards NLU: On meaning, form, and understanding in the age of data”. In: *Proc. of ACL*.

- Bengio, Yoshua et al. (2003). “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.
- Berant, Jonathan et al. (2013). “Semantic parsing on freebase from question-answer pairs”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1533–1544.
- Bhatnagar, Shalabh et al. (2009). “Natural actor–critic algorithms”. In: *Automatica* 45.11, pp. 2471–2482.
- Billard, Aude et al. (2008). *Survey: Robot programming by demonstration*. Tech. rep. Springer.
- Bisk, Yonatan, Deniz Yuret, and Daniel Marcu (2016). “Natural language communication with robots”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 751–761.
- Branavan, S.R.K. et al. (Aug. 2009). “Reinforcement Learning for Mapping Instructions to Actions”. In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, pp. 82–90. URL: <https://www.aclweb.org/anthology/P09-1010>.
- Celemin, Carlos and Javier Ruiz-del Solar (2019). “An interactive framework for learning continuous actions policies based on corrective feedback”. In: *Journal of Intelligent & Robotic Systems* 95.1, pp. 77–97.
- Celemin, Carlos et al. (2019). “Reinforcement learning of motor skills using Policy Search and human corrective advice”. In: *The International Journal of Robotics Research* 38.14, pp. 1560–1580.
- Chernova, Sonia and Andrea L Thomaz (2014). “Robot learning from human teachers”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8.3, pp. 1–121.
- Cho, Kyunghyun et al. (2014a). “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078*.
- Cho, Kyunghyun et al. (2014b). “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259*.
- Christiano, Paul et al. (2017). “Deep reinforcement learning from human preferences”. In: *Advances in Neural Information Processing Systems*, pp. 4302–4310.
- Clarke, James et al. (2010). “Driving semantic parsing from the world’s response”. In: *Proceedings of the fourteenth conference on computational natural language learning*, pp. 18–27.
- Cruz, Francisco et al. (2016). “Training agents with interactive reinforcement learning and contextual affordances”. In: *IEEE Transactions on Cognitive and Developmental Systems* 8.4, pp. 271–284.
- Daumé III, Hal (2009). “Frustratingly easy domain adaptation”. In: *arXiv preprint arXiv:0907.1815*.
- Davis, Sean and Paul S Meltzer (2007). “GEOquery: a bridge between the Gene Expression Omnibus (GEO) and BioConductor”. In: *Bioinformatics* 23.14, pp. 1846–1847.
- De Alfaro, Luca, Thomas A Henzinger, and Rupak Majumdar (2003). “Discounting the future in systems theory”. In: *Automata, Languages and Programming*. Springer, pp. 1022–1037.
- De Winter, Joris et al. (2019). “Accelerating Interactive Reinforcement Learning by Human Advice for an Assembly Task by a Cobot”. In: *Robotics* 8.4, p. 104.
- Dong, Li and Mirella Lapata (2016). “Language to logical form with neural attention”. In: *arXiv preprint arXiv:1601.01280*.
- Fard, Seyed Mehdi Hazrati, Ali Hamzeh, and Sattar Hashemi (2013). “Using reinforcement learning to find an optimal set of features”. In: *Computers & Mathematics with Applications* 66.10, pp. 1892–1904.
- Ghadirzadeh, Ali et al. (2020). “Human-centered collaborative robots with deep reinforcement learning”. In: *arXiv preprint arXiv:2007.01009*.



- Graves, Alex and Navdeep Jaitly (2014). “Towards end-to-end speech recognition with recurrent neural networks”. In: *International conference on machine learning*, pp. 1764–1772.
- Griffith, Shane et al. (2013). “Policy shaping: Integrating human feedback with reinforcement learning”. In: *Advances in Neural Information Processing Systems*, pp. 2625–2633.
- Hadfield-Menell, Dylan et al. (2016). “Cooperative inverse reinforcement learning”. In: *Advances in Neural Information Processing Systems*, pp. 3909–3917.
- Ho, Mark K et al. (2015). “Teaching with rewards and punishments: Reinforcement or communication?” In: *Proceedings of the 37th Annual Meeting of the Cognitive Science Society*.
- Ho, Mark K. et al. (2017). “Social is special: A normative framework for teaching with and learning from evaluative feedback”. In: *Cognition* 167, pp. 91–106.
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.
- Hussein, Ahmed et al. (2017). “Imitation learning: A survey of learning methods”. In: *ACM Computing Surveys (CSUR)* 50.2, pp. 1–35.
- Isbell, Charles et al. (2001). “A social reinforcement learning agent”. In: *Proceedings of the fifth international conference on Autonomous agents*. ACM, pp. 377–384.
- Isbell, Charles Lee et al. (2006). “Cobot in LambdaMOO: An adaptive social statistics agent”. In: *Autonomous Agents and Multi-Agent Systems* 13.3, pp. 327–354.
- Iyer, Srinivasan et al. (2017). “Learning a neural semantic parser from user feedback”. In: *arXiv preprint arXiv:1704.08760*.
- Johnson, Tim (1984). “Natural language computing: the commercial applications”. In: *The Knowledge Engineering Review* 1.3, pp. 11–23.
- Kalchbrenner, Nal and Phil Blunsom (2013). “Recurrent continuous translation models”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709.
- Kamath, Aishwarya and Rajarshi Das (2018). “A survey on semantic parsing”. In: *arXiv preprint arXiv:1812.00978*.
- Kasenberg, Daniel and Matthias Scheutz (2017). “Interpretable Apprenticeship Learning with Temporal Logic Specifications”. In: *Proceedings of the 56th IEEE Conference on Decision and Control*.
- Kassner, Nora and Hinrich Schütze (2020). “Negated and Misprimed Probes for Pretrained Language Models: Birds Can Talk, But Cannot Fly”. In: Association for Computational Linguistics.
- Kearns, Michael and Satinder Singh (1998). “Near-optimal reinforcement learning in polynomial time”. In: *Proceedings of the 15th International Conference on Machine Learning*, pp. 260–268. URL: [citeseer.nj.nec.com/kearns98nearoptimal.html](http://citeseer.nj.nec.com/kearns98nearoptimal.html).
- Kim, Su Kyoung et al. (2017). “Intrinsic interactive reinforcement learning—Using error-related potentials for real world human-robot interaction”. In: *Scientific reports* 7.1, pp. 1–16.
- Klingspor, Volker, John Demiris, and Michael Kaiser (1997). “Human-robot communication and machine learning”. In: *Applied Artificial Intelligence* 11.7, pp. 719–746.
- Knox, W Bradley and Peter Stone (2009a). “Interactively shaping agents via human reinforcement: The TAMER framework”. In: *Proceedings of the fifth international conference on Knowledge capture*. ACM, pp. 9–16.
- (2009b). “Interactively shaping agents via human reinforcement: The TAMER framework”. In: *Proceedings of the Fifth International Conference on Knowledge Capture*, pp. 9–16.
- (2013). “Learning non-myopically from human-generated reward”. In: *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, pp. 191–202.
- Knox, W Bradley, Peter Stone, and Cynthia Breazeal (2013). “Training a robot via human feedback: A case study”. In: *Social Robotics*. Springer, pp. 460–470.

- Knox, W Bradley et al. (2012). “How humans teach agents”. In: *International Journal of Social Robotics* 4.4, pp. 409–421.
- Knox, William Bradley (2012). “Learning from human-generated reward”. PhD thesis. University of Texas at Austin.
- Kollar, Thomas, Jayant Krishnamurthy, and Grant P Strimel (2013). “Toward Interactive Grounded Language Acquisition.” In: *Robotics: Science and systems*. Vol. 1, pp. 721–732.
- Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press.
- Krening, Samantha and Karen M Feigh (2019). “Effect of interaction design on the human experience with interactive reinforcement learning”. In: *Proceedings of the 2019 on Designing Interactive Systems Conference*, pp. 1089–1100.
- Kress-Gazit, H., G.E. Fainekos, and G.J. Pappas (2009). “Temporal-Logic-Based Reactive Mission and Motion Planning”. In: *IEEE Tans. on Robotics* 25, pp. 1370–1381.
- Krishnamurthy, Jayant, Pradeep Dasigi, and Matt Gardner (2017). “Neural semantic parsing with type constraints for semi-structured tables”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1516–1526.
- Krishnamurthy, Jayant and Tom Mitchell (2012). “Weakly supervised training of semantic parsers”. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 754–765.
- “Kuri Robot Website”. In: (). URL: <https://www.heykuri.com/explore-kuri/#feature-cap-touch-sensors>.
- Lahijanian, M., S. B. Andersson, and C. Belta (2011). “Control of Markov decision processes from PCTL specifications”. In: *Proc. of the American Control Conference*, pp. 311–316.
- León, Adrián et al. (2011). “Teaching a Robot to Perform Task through Imitation and On-line Feedback”. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Ed. by César San Martin and Sang-Woon Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 549–556. ISBN: 978-3-642-25085-9.
- Li, G. et al. (2019). “Human-Centered Reinforcement Learning: A Survey”. In: *IEEE Transactions on Human-Machine Systems* 49.4, pp. 337–349.
- Li, Guangliang et al. (2020). “Facial feedback for reinforcement learning: a case study and offline analysis using the TAMER framework”. In: *Autonomous Agents and Multi-Agent Systems* 34.1, pp. 1–29.
- Liang, Chen et al. (2016). “Neural symbolic machines: Learning semantic parsers on freebase with weak supervision”. In: *arXiv preprint arXiv:1611.00020*.
- Littman, Michael L. (2015). “Reinforcement learning improves behaviour from evaluative feedback”. In: *Nature* 521.7553, pp. 394–556.
- Littman, Michael L. et al. (2017). “Environment-Independent Task Specifications via GLTL”. arXiv preprint arXiv:1704.04341.
- Loftin, Robert et al. (2014). “A Strategy-Aware Technique for Learning Behaviors from Discrete Human Feedback”. In: *Proceedings of the Twenty-Eighth Association for the Advancement of Artificial Intelligence Conference*.
- Loftin, Robert et al. (2015). “Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning”. In: *Autonomous Agents and Multi-Agent Systems* 30.1, pp. 30–59.
- Loftin, Robert T. et al. (2016). “Learning behaviors via human-delivered discrete feedback: Modeling implicit feedback strategies to speed up learning”. In: *Autonomous Agents and Multi-Agent Systems* 30.1, pp. 30–59.
- MacGlashan, James et al. (2017). “Interactive Learning from Policy-Dependent Human Feedback”. In: *Proceedings of the Thirty-Fourth International Conference on Machine Learning*.

- MacMahon, Matt, Brian Stankiewicz, and Benjamin Kuipers (2006). “Walk the talk: Connecting language, knowledge, and action in route instructions”. In: *Def 2.6*, p. 4.
- Manna, Zohar and Amir Pnueli (1992). *The Temporal Logic of Reactive & Concurrent Sys.* Springer.
- Matthews, David and Josh Bongard (2020). “Crowd grounding: finding semantic and behavioral alignment through human robot interaction.” In: *Artificial Life Conference Proceedings*. MIT Press, pp. 148–156.
- Matuszek, Cynthia et al. (2014). “Learning from unscripted deictic gesture and language for human-robot interactions”. In: *UMBC Faculty Collection*.
- Mavridis, Nikolaos (2015). “A review of verbal and non-verbal human–robot interactive communication”. In: *Robotics and Autonomous Systems* 63, pp. 22–35.
- Mehta, Nikhil and Dan Goldwasser (2019). “Improving Natural Language Interaction with Robots Using Advice”. In: *arXiv preprint arXiv:1905.04655*.
- Mudgal, Sidharth et al. (2018). “Deep learning for entity matching: A design space exploration”. In: *Proceedings of the 2018 International Conference on Management of Data*, pp. 19–34.
- Ng, Andrew Y. and Stuart Russell (2000). “Algorithms for inverse reinforcement learning”. In: *International Conference on Machine Learning*, pp. 663–670.
- Ngo Anh Vien and W. Ertel (2012). “Reinforcement learning combined with human feedback in continuous state and action spaces”. In: *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pp. 1–6.
- Ni, Pin et al. (2020). “Natural language understanding approaches based on joint task of intent detection and slot filling for IoT voice interaction”. In: *Neural Computing and Applications*, pp. 1–18.
- Pasupat, Panupong and Percy Liang (2015). “Compositional semantic parsing on semi-structured tables”. In: *arXiv preprint arXiv:1508.00305*.
- Patel, R., Ellie Pavlick, and Stefanie Tellex (2020). “Grounding Language to Non-Markovian Tasks with No Supervision of Task Specifications”. In:
- Patki, Siddharth et al. (2019). “Inferring compact representations for efficient natural language understanding of robot instructions”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6926–6933.
- Peng, Bei et al. (2017). “Curriculum Design for Machine Learners in Sequential Decision Tasks”. In: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*.
- Pilarski, Patrick M et al. (2011). “Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning”. In: *2011 IEEE International Conference on Rehabilitation Robotics*. IEEE, pp. 1–7.
- Poon, Hoifung and Pedro Domingos (2009). “Unsupervised semantic parsing”. In: *Proceedings of the 2009 conference on empirical methods in natural language processing*, pp. 1–10.
- Ramesh, Divya et al. (2020). “Yesterday’s Reward is Today’s Punishment: Contrast Effects in Human Feedback to Reinforcement Learning Agents”. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 1090–1097. ISBN: 9781450375184.
- Rivest, Ronald L and Robert Sloan (1994). “A formal model of hierarchical concept-learning”. In: *Information and Computation* 114.1, pp. 88–114.
- Roesler, Oliver and Ann Nowé (2019). “Action learning and grounding in simulated human–robot interactions”. In: *The Knowledge Engineering Review* 34.
- Roesler, Oliver et al. (2019). “Evaluation of word representations in grounding natural language instructions through computational human-robot interaction”. In: *2019 14th*

- ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, pp. 307–316.
- Roy, Deb K (2002). “Learning visually grounded words and syntax for a scene description task”. In: *Computer speech & language* 16.3-4, pp. 353–385.
- Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: *nature* 323.6088, pp. 533–536.
- Russell, Stuart J. and Peter Norvig (1994). *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall. ISBN: 0-13-103805-2.
- Schalkwyk, Johan et al. (2010). ““your word is my command”: Google search by voice: A case study”. In: *Advances in speech recognition*. Springer, pp. 61–90.
- Schuster, Mike and Kuldip K Paliwal (1997). “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11, pp. 2673–2681.
- Singh, S, R L Lewis, and A G Barto (2009). “Where do rewards come from?” In: *Proceedings of the Annual Conference of the Cognitive Science Society*.
- Steels, Luc and Paul Vogt (1997). “Grounding adaptive language games in robotic agents”. In: *Proceedings of the fourth european conference on artificial life*. Vol. 97.
- Suay, H. B. and S. Chernova (2011). “Effect of human guidance and state space size on Interactive Reinforcement Learning”. In: *2011 RO-MAN*, pp. 1–6.
- Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*, pp. 3104–3112.
- Sutton, Richard S. and Andrew G. Barto (1998b). *Reinforcement Learning: An Introduction*. The MIT Press.
- Sutton, Richard S and Andrew G Barto (1998a). *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge.
- Sutton, Richard S et al. (1999). “Policy Gradient Methods for Reinforcement Learning with Function Approximation.” In: *NIPS*. Vol. 99, pp. 1057–1063.
- Tabrez, Aaquib and Bradley Hayes (2019). “Improving human-robot interaction through explainable reinforcement learning”. In: *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, pp. 751–753.
- Tan, Hao and Mohit Bansal (2017). “Source-target inference models for spatial instruction understanding”. In: *arXiv preprint arXiv:1707.03804*.
- Tenorio-Gonzalez, Ana C, Eduardo F Morales, and Luis Villaseñor-Pineda (2010). “Dynamic reward shaping: training a robot by voice”. In: *Advances in Artificial Intelligence—IBERAMIA 2010*. Springer, pp. 483–492.
- Thomason, Jesse et al. (2019). “Improving grounded natural language understanding through human-robot dialog”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6934–6941.
- Thomaz, Andrea L and Cynthia Breazeal (2007). “Robot learning via socially guided exploration”. In: *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*. IEEE, pp. 82–87.
- (2008). “Teachable robots: Understanding human teaching behavior to build more effective robot learners”. In: *Artificial Intelligence* 172, pp. 716–737.
- Thomaz, Andrea Lockerd and Cynthia Breazeal (2006). “Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance”. In: *AAAI*. Vol. 6, pp. 1000–1005.
- Thompson, Henry S et al. (1993). “The HCRC map task corpus: natural dialogue for speech recognition”. In: *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop Held at Plainsboro, New Jersey, March 21-24, 1993*.
- Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Advances in neural information processing systems*, pp. 5998–6008.

- Vien, Ngo Anh, Wolfgang Ertel, and Tae Choong Chung (2013). "Learning via human feedback in continuous state and action spaces". In: *Applied intelligence* 39.2, pp. 267–278.
- Vinyals, Oriol et al. (2015). "Grammar as a foreign language". In: *Advances in neural information processing systems*, pp. 2773–2781.
- Wang, Guan et al. (2020). "Teaching a Robot Tasks of Arbitrary Complexity via Human Feedback". In: *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pp. 649–657.
- Wang, Yushi, Jonathan Berant, and Percy Liang (2015). "Building a semantic parser overnight". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1332–1342.
- Warnell, Garrett et al. (2017). "Deep tamer: Interactive agent shaping in high-dimensional state spaces". In: *arXiv preprint arXiv:1709.10163*.
- Watkins, Christopher J. C. H. (1989). "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College.
- Watkins, Christopher J. C. H. and Peter Dayan (1992). "Q-Learning". In: *Machine Learning* 8.3, pp. 279–292.
- Winograd, Terry (1980). "What does it mean to understand language?" In: *Cognitive science* 4.3, pp. 209–241.
- Wirth, Christian and Johannes Fürnkranz (2013). "Preference-based reinforcement learning: A preliminary survey". In: *Proceedings of the ECML/PKDD-13 Workshop on Reinforcement Learning from Generalized Feedback: Beyond Numeric Rewards*.
- Wolff, Eric M., Ufuk Topcu, and Richard M. Murray (2012). "Robust Control of Uncertain Markov Decision Processes with Temporal Logic Specifications". In: *Proc. of the IEEE Conference on Decision and Control*.
- Wong, Yuk Wah and Raymond Mooney (2006). "Learning for semantic parsing with statistical machine translation". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pp. 439–446.
- Wongpiromsarn, T., U. Topcu, and R.M. Murray (2012). "Receding Horizon Temporal Logic Planning". In: *IEEE T. on Automatic Control* 57, pp. 2817–2830.
- Woods, William A (1973). "Progress in natural language understanding: an application to lunar geology". In: *Proceedings of the June 4-8, 1973, national computer conference and exposition*, pp. 441–450.
- Yu, Chao et al. (2020). "Interactive RL via Online Human Demonstrations". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2065–2067.
- Zelle, John M and Raymond J Mooney (1996). "Learning to parse database queries using inductive logic programming". In: *Proceedings of the national conference on artificial intelligence*, pp. 1050–1055.
- Zettlemoyer, Luke S and Michael Collins (2012). "Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars". In: *arXiv preprint arXiv:1207.1420*.
- Zhang, Wen et al. (2019). "Bridging the gap between training and inference for neural machine translation". In: *arXiv preprint arXiv:1906.02448*.
- Zheng, Weiguo et al. (2019). "Interactive natural language question answering over knowledge graphs". In: *Information Sciences* 481, pp. 141–159.
- Ziebart, Brian D et al. (2008). "Maximum Entropy Inverse Reinforcement Learning". In: *AAAI*. Vol. 8, pp. 1433–1438.