The Dissertation Committee for Prasoon Goyal
certifies that this is the approved version of the following dissertation:

Using Natural Language to Aid Task Specification in Sequential Decision Making Problems

Committee:

Raymond J. Mooney, Co-Supervisor

Scott D. Niekum, Co-Supervisor

Peter Stone

Yoav Artzi

Using Natural Language to Aid Task Specification in Sequential Decision Making Problems

by

Prasoon Goyal

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2022

Acknowledgments

First and foremost, I would like to thank my advisors Raymond J. Mooney (Ray) and Scott Niekum for their invaluable guidance throughout my Ph.D. journey. From brainstorming ideas, to discussing short-term and long-term research directions, my meetings with them were instrumental in the final work that I have been able to produce over the last 5 years. I deeply admire Ray's long experience in the field, which has given me an interesting perspective through our discussions. Further, Ray would often bring up connections between my current research ideas and ideas that had been tried in the past, giving me pointers that I might not have come across on my own. My interactions with Scott, especially with his eclectic research interests spanning theoretical reinforcement learning to robotic applications, took me from having no background in reinforcement learning to becoming a reasonably proficient researcher in the field. Contrary to the countless stories I had read of how a Ph.D. is a stressful experience, working with Ray and Scott made my experience extremely enjoyable and intellectually satisfying. I would also like to thank Peter Stone and Yoav Artzi for being on my committee, and for their insightful suggestions and questions, which have helped improve my dissertation.

Next, I would like to thank my fellow graduate students, with whom I could discuss the intricate details of my research, as well as from whom I learned about various other research problems outside of my area. This includes Ishan Durugkar, Haresh Karnan, Aishwarya Padmakumar, Harshit Sikchi, Daniel Brown, Yuchen Cui, Ajinkya Jain,

Akanksha Saran, Sheena Panthaplackel, Jialin Wu, Albert Yu, Vanya Cohen, Wonjoon Goo, Caleb Chuck, Jordan Schneider, Christina Yuan, Jesse Thomason, Santhosh Ramakrishnan, Tushar Nagarajan, Stephen Giguere, and Rudolf Lioutikov. Of course, many of these were also among my closest friends in Austin, and I'll cherish the time spent with them, especially all the lunch group meetings.

I also learned a lot about different areas of AI from other faculty members, particularly through reading group meetings, including Greg Durrett, David Harwath, Yu Chen, Eunsol Choi, Katrin Erk, and Jessy Li. My interactions with mentors during my various internships have also been very enriching, and have played a part in who I am today as a researcher.

While I learned a lot from the aforementioned people during my Ph.D. journey, I have been fortunate to have a great set of mentors well before I started my Ph.D. During my MS at NYU, I was fortunate to have got a chance to work with Mehryar Mohri and Corinna Cortes, which was a great experience. Prior to that, during my undergraduate program at IIT Delhi, it was a very rewarding experience to work with Parag Singla sir. Not only was working with him intellectually satisfying, but his guidance was also instrumental for me in making crucial decisions about graduate school. I would also like to thank other professors at IIT Delhi whom I got a chance to work with or learn from, including Prof. Naveen Garg, Prof. Jayadeva, Prof. Manik Varma, and Prof. Subhashis Banerjee. Next, I would like to thank my teachers at Vidyamandir Classes, in particular Brijmohan Gupta, Shyam Mohan Gupta, and Manmohan Gupta – in addition to their guidance in preparing me for the IIT JEE, their insights on life have motivated me through the years. Finally, I would like to thank my teachers at The Mother's International School, particularly Dash sir for developing

my initial interest in computer science, and Pankaj Vajpayee sir for instilling in me a quest for learning.

Outside of work, I was lucky to have a set of wonderful friends, who made my time fun and enjoyable. Other than those already mentioned, I'd like to thank Rohit Satija, Ram Reddy, Ankur Agarwal, Caleb Phillips, Christian Hogan, Aryaman Samyal, Nikhil Raj, and Shubhankar Agarwal. I am also grateful to the Prem Milan community here in Austin, for all the wonderful gatherings involving games and deep conversations.

Keeping in touch with a lot of friends outside Austin over the phone or meeting them when I visited India was also quite refreshing, and helped me keep working towards my goal. In particular, I'd like to thank Pulin Agrawal and Sanjit Singh Batra.

I would also like to thank the UTCS staff, in particular Stacy Miller, Katie Traughber, Megan Booth, Josh Byrnes, Matt Larson, Amy Bush, and Kay Nettle for helping with all the administrative work, or systems-related issues.

Last but not the least, I'd like to thank my parents Krishna Mohan Goyal and Prabha Goyal, and my sister Kanika Goyal. They supported me in my somewhat unusual interests and choices from the very beginning and encouraged me at every step in the process. I would not be here without their belief in me.

Prasoon Goyal

The University of Texas at Austin
July 2022

Using Natural Language to Aid Task Specification in Sequential Decision Making Problems

by

Prasoon Goyal, Ph.D.
The University of Texas at Austin, 2022

Supervisors: Raymond J. Mooney
Scott D. Niekum

Building intelligent agents that can help humans accomplish everyday tasks, such as a personal robot at home or a robot in a work environment, is a long-standing goal of artificial intelligence. One of the requirements for such general-purpose agents is the ability to teach them new tasks or skills relatively easily. Common approaches to teaching agents new skills include reinforcement learning (RL) and imitation learning (IL). However, specifying the task to the learning agent, i.e. designing effective reward functions for reinforcement learning and providing demonstrations for imitation learning, are often cumbersome and time-consuming. Further, designing reward functions and providing a set of demonstrations that sufficiently disambiguates the desired task may not be particularly accessible for end users without a technical background.

In this dissertation, we explore using natural language as an auxiliary signal to aid task specification, which reduces the burden on the end user. To make reward design easier,

we propose a novel framework that is used to generate language-based rewards in addition to the extrinsic rewards from the environment for faster policy training using RL. We show that using our framework, very simple extrinsic rewards along with a natural language description of the task are sufficient to teach new tasks to the learning agent. To ameliorate the problem of providing demonstrations, we propose a new setting that enables an agent to learn a new task without demonstrations in an IL setting, given a demonstration from a related task and a natural language description of the difference between the desired task and the demonstrated task. The techniques we develop for this setting would enable teaching multiple related tasks to learning agents by providing a small set of demonstrations and several natural language descriptions, thereby reducing the burden of providing demonstrations for each task.

The primary contributions of this dissertation include novel problem settings, benchmarks, and algorithms that allow using natural language as an auxiliary modality for task specification in RL and IL. We believe this dissertation will serve as a foundation for future research along these lines, to make progress toward having intelligent agents that can conveniently be taught new tasks by end users.

Table of Contents

Acknowledgments **iv**

Abstract **vii**

List of Tables **xiii**

List of Figures **xiv**

Chapter 1. Introduction **1**

1.1 Motivation 3

1.1.1 Reinforcement Learning 3

1.1.2 Imitation Learning 5

1.1.3 Other Related Problems 6

1.2 Contributions and Dissertation Outline 8

Chapter 2. Background and Related Work **11**

2.1 Sequential Decision Making 11

2.1.1 Terminology 11

2.1.2 Reinforcement Learning 13

2.1.3 Imitation Learning 17

2.2 Natural Language Processing 20

2.3 Language Grounding 24

2.3.1 Grounding Language to Images and Videos 24

2.3.2 Instruction-following 24

2.3.3 Language to Aid Learning 26

Chapter 3. Using Natural Language for Reward Shaping in Reinforcement Learning 29
3.1 Introduction . . . 29
3.2 Approach . . . 32
3.2.1 LanguagE-Action Reward Network . . . 34
3.2.2 Using Language-based Rewards in RL . . . 36
3.3 Domain and Dataset . . . 39
3.4 Experimental Evaluation . . . 42
3.4.1 How Much Does Language Help? . . . 43
3.4.2 Analysis of Language-based Rewards . . . 47
3.4.3 Sensitivity Analysis . . . 50
3.5 Conclusions . . . 50

Chapter 4. Guiding Reinforcement Learning Using Natural Language by Mapping Pixels to Rewards 52
4.1 Introduction . . . 52
4.2 Approach . . . 54
4.2.1 PixL2R: Pixels and Language to Reward . . . 55
4.2.2 Policy Learning Phase . . . 60
4.3 Domain and Dataset . . . 60
4.4 Experiments . . . 63
4.4.1 Training the PixL2R model . . . 63
4.4.2 Policy Training with Language-based Rewards . . . 66
4.4.3 Ablations . . . 69
4.4.4 Word-level Analysis . . . 73
4.5 Conclusions . . . 74

Chapter 5. Zero-shot Task Adaptation Using Natural Language 76
5.1 Introduction . . . 76
5.2 Problem Definition . . . 80
5.3 LARVA: Language-Aided Reward and Value Adaptation . . . 81
5.3.1 Network Architecture . . . 81
5.3.1.1 Target Goal Predictor . . . 83

5.3.1.2 Reward / Value Network 84
5.3.2 Training 84
5.4 Environment and Dataset 85
5.4.1 The Organizer Environment 86
5.4.2 Language Data 87
5.5 Experiments 90
5.5.1 Ablations 93
5.5.2 Compositionality 94
5.5.3 Amount of Data Needed 95
5.6 Conclusions 96

Chapter 6. Relational Language-guided Task Adaptation for Imitation Learning 97
6.1 Introduction 97
6.2 Benchmark Datasets 100
6.3 Relational Reward Adaptation: Approach 105
6.3.1 Goal Prediction 106
6.3.2 Distance Function Learning 108
6.3.3 Training 108
6.4 Relational Reward Adaptation: Experiments 110
6.4.1 Details about the setup 110
6.4.2 Results 110
6.5 Relational Policy Adaptation: Approach 113
6.6 Relational Policy Adaptation: Experiments 115
6.7 Qualitative Analysis 117
6.8 Combining Reward and Policy Adaptation 118
6.9 Conclusions 122

Chapter 7. Future Work 124
7.1 Short-term Future Directions 124
7.2 Long-term Future Directions 126
7.2.1 Richer Tasks and Evaluation 126
7.2.2 Experiments on Real Robots 127

7.2.3 Reward Shaping for Multi-step Tasks 128
7.2.4 Task Adaptation for a Broader Set of Tasks 130
7.2.5 Task Adaptation with Multiple Source Tasks 132
7.2.6 Language-aided Imitation Learning 134

Chapter 8. Conclusion **136**

Bibliography **141**

List of Tables

3.1 Examples of descriptions collected using Amazon Mechanical Turk 42
3.2 Analysis of language-based rewards 47

4.1 Examples of descriptions collected using AMT. 64
4.2 Comparison of various ablations to the Dense+RGR model. 71
4.3 Average magnitude of gradients for different words in a description for the relatedness score prediction. 74

5.1 Types of adaptations used in our experiments. For each type, an example of source and target tasks is shown. 88
5.2 Examples of template-generated and natural language descriptions collected using AMT. 89
5.3 Success rates of different models 92
5.4 Success rates (%) when using varying amounts of synthetic and natural language data to train LARVA. The row labels show the number of natural language examples used while the column labels show the number of synthetic language examples used. 95

6.1 Examples of template-generated and natural language descriptions collected using AMT. 104
6.2 Success rates for different models on Room Rearrangement and Room Navigation domains. We report both the raw success rates (unnormalized), and success rates normalized by the oracle setting performance. 111
6.3 Success rates for relational policy adaptation on the Room Rearrangement and Room Navigation domains. 116
6.4 No. of successes when reward adaptation is combined with policy adaptation. 122

List of Figures

3.1 An agent exploring randomly to complete the task described by the blue trajectory may need considerable amount of time to learn the behavior. By giving natural language instructions like "Jump over the skull while going to the left", we can give intermediate signals to the agent for faster learning. . . 30

3.2 Our framework consists of the standard RL module containing the agent-environment loop, augmented with a LanguagE-Action Reward Network (LEARN) module. 33

3.3 Neural network architecture for LEARN (Section 3.2.1) 36

3.4 Sample Mechanical Turk task for collecting natural language descriptions. . . 41

3.5 Comparison of different reward functions: The solid lines represent the mean successful episodes averaged over all tasks, and the shaded regions represent 95% confidence intervals. 45

3.6 Comparisons of different reward functions for selected tasks 48

3.7 Effect of adding noise to the predictions of LEARN. 50

4.1 A simulated robot completing a task ("push the green button") in the Meta-World domain. 53

4.2 Viewpoints used for data collection and experiments. 56

4.3 Neural network architecture: The sequence of frames from the three viewpoints are passed through three separate CNN feature extractors, and then concatenated across views. The sequence is then passed through an LSTM to obtain an encoding of the trajectory. The given linguistic description is encoded using a randomly initialized embedding layer followed by an LSTM. The outputs of the two LSTMs is concatenated and passed through a sequence of 2 linear layers to generate the final prediction. 57

4.4 List of objects used 62

4.5 Distribution of number of words per description in the collected dataset. . . 65

4.6 A comparison of policy training curves for different reward models. The shaded regions denote 95% confidence intervals. 68

5.1 Example of the setting: The top row shows the *source task*, while the bottom shows the *target task*. Given a demonstration of the source task, and a natural language description of the difference between the two tasks such as "In the third step, move the green flat block from bottom left to top left.", our goal is to train an agent to perform the target task *without* any demonstrations of the target task. 77
5.2 Neural network architecture for LARVA 82
5.3 Objects in the Organizer Environment 86
5.4 Interface for collecting paraphrases using Amazon Mechanical Turk 87

6.1 Example of the setting: The left image shows a demonstration of the source task, where the red point is the initial location of the agent, and the green point is the final location. The image on the right shows the target task, with the desired goal location marked with the green 'x'. The objective is to train an agent to perform the target task without any demonstrations of the target task, which requires reasoning about relative positions of entities. 98
6.2 We present two independent approaches to learn a target task policy – relational reward adaptation (Section 6.3) and relational policy adaptation (Section 6.5). In Section 6.8, we show how to combine these two approaches. . . 99
6.3 Adaptations used in the Room Rearrangement (top) and Room Navigation (bottom) domains. 102
6.4 Neural Network architecture for relational goal prediction. 106
6.5 Policy Adaptation Approach 114
6.6 Visualization of predicted goal for two test datapoints. The yellow X denotes the goal position under the source task, and the red and blue X's denote the predicted and true goal positions under the target task. 117
6.7 Visualization of predicted goal for two test datapoints. The red X denotes the initial position of the agent, the yellow X denotes the true goal position under the source task, and the blue X denotes the true goal position under the target task. 118
6.8 Initializing the value and policy networks of the actor-critic model using the reward adaptation and policy adaptation approaches. 119
6.9 Learning curves comparing the policy training curves on target tasks when using uninitialized PPO networks and PPO networks initialized using policy adaptation, on the Rearrangement (left) and Navigation (right) domains. . . 122

Chapter 1

Introduction

With the extraordinary progress in the field of artificial intelligence (AI) over the last several decades, we are closer than ever to the holy grail of the field—having intelligent agents around us to help with everyday tasks. These agents could be deployed in home environments to assist humans with tasks such as cooking, cleaning, house maintenance, and personalized care, or in work environments for tasks such as manufacturing, packaging, inventory management, transportation of goods, and tasks that are dangerous for humans. Since the list of tasks that we would like the agents to complete is potentially endless, one of the key desiderata in building such general-purpose agents is to endow them with the ability to learn new tasks from humans. Importantly, these methods should require minimal human effort for them to be practical. Moreover, since these agents would work alongside humans who are largely not AI experts, it is crucial that the methods to teach new tasks to agents are amenable for use by non-experts.

A large fraction of everyday tasks can be modeled using the **sequential decision making** framework, which involves an agent interacting with an environment. At each step, the agent observes the state of the environment, and takes an action, resulting in a change in the state of the environment. The agent continues to take actions until the desired task is completed. Over the past few decades, several approaches have been developed to train

agents to perform a variety of tasks in this setting. Broadly, these approaches can be divided into two classes—**reinforcement learning** (RL), and **imitation learning** (IL). The agent's behavior is usually represented using a function – known as the *policy* – that takes a state as input and outputs the action that the agent takes at this state. In reinforcement learning, when the agent takes an action leading to a change in the state of the environment, it receives a numeric score, known as the *reward*, from the environment. The objective for the agent is to execute a sequence of actions that maximize the sum of rewards received, that is, learn a policy that obtains the maximum sum of rewards. In imitation learning, also known as *learning from demonstrations*, the agent is provided with demonstrations of the desired task. The agent needs to infer the demonstrator's intent and learn a policy that completes the demonstrator's intended task.

Thus, from the user's perspective, in reinforcement learning, the intended task is specified to the agent by designing a reward function, maximizing which completes the task, and in imitation learning, the intended task is specified to the agent by providing demonstrations. As we discuss in Section 1.1, both these task specification modalities are cumbersome for end users, and therefore, techniques that aid task specification are required to make these systems ready for the real world.

A promising direction to make task specification more amenable for end users is to use natural language as an auxiliary signal since language has evolved alongside the human race to be a convenient and flexible form of conveying intentions. Thus, in this dissertation, we seek to answer the following question: **How can natural language be used as an auxiliary signal to reduce the burden of task specification on the end user for sequential decision making problems?**

1.1 Motivation

1.1.1 Reinforcement Learning

As described above, a reward function is used to specify the intended task in reinforcement learning. The reward function must, of course, be such that executing a sequence of actions that completes the desired task results in the maximum sum of rewards. Additionally, we would like the reward function to *guide* the agent towards the goal, by giving it a high reward for progress towards the goal, a low reward for no progress, and an even lower reward for actions that take it further away from the goal. This results in a fundamental trade-off between the ease of designing the reward function (for the user) and the ease of learning from the reward function (for the agent). On one end of the spectrum, we can have a reward function that is a non-zero positive value at the goal state, and zero at all the other states. Such rewards are known as *sparse* reward functions. More generally, for sparse reward functions, the distribution of reward values over all the states in the environment has entropy close to zero. For instance, in a cooking task, a sparse reward function would have a value of 1 (or any other positive value) when the intended dish has been successfully cooked, and a value of zero in all other states. The agent's sum of rewards is zero for all sequences of actions that do not complete the task, and is one for all sequences of actions that successfully complete the task. Thus, maximizing the sum of rewards results in the successful completion of the task. Such reward functions are easy to design for the user but are difficult for the agent to learn from, since the reward function doesn't guide the agent towards the goal—an agent with no other knowledge of the intended task must randomly execute sequences of actions until it accidentally executes a successful sequence, at which point it receives a useful signal about the task. This could require a significant amount

of exploration by the agent before it learns the task, and might become infeasible as the complexity of the task grows. Thus, sparse reward functions are easy to design for the user but difficult to learn from for the agent. On the other end of the spectrum, we have carefully designed reward functions that provide a non-zero signal at each state, such that better actions result in higher rewards compared to worse actions. These reward functions are known as *dense* reward functions. For instance, in a cooking task, a dense reward function will have non-zero values for each step. Further, if a step requires, say, heating the ingredients for 5 minutes, then a dense reward function may be defined for this step that is a function of the heating time—heating for exactly 5 minutes results in the maximum reward at this step, while heating for a shorter or longer duration results in a lower reward. The agent can then learn the desired task efficiently since it receives a useful signal at each step, which allows it to explore promising action sequences more effectively. However, designing such intricate reward functions requires a considerable amount of effort by the user. Thus, fine-grained dense reward functions are easy to learn from for the agent but difficult to design for the user.

In the first part of this dissertation (Chapters 3 and 4), we develop techniques that use natural language to get the best of both worlds – the ease of designing, as in sparse reward functions, and the ease of learning, as in fine-grained dense reward functions – by generating auxiliary rewards from a natural language description of the task that are used in conjunction with sparse or coarse-grained dense rewards.

1.1.2 Imitation Learning

In imitation learning, from the user's perspective, the intended task is specified to the agent by providing demonstrations. A demonstration shows a sequence of actions that successfully completes the desired task. However, providing demonstrations is often cumbersome due to various reasons. First, a single demonstration may not be enough to uniquely specify the desired task. For instance, if the demonstration shows moving a mug from the kitchen counter to the dining table, the intended task could be to clear the kitchen counter (thus any objects from the kitchen counter should be moved to the table), or to set up a mug on the dining table (thus if the mug were in the dishwasher, it should be moved from the dishwasher to dining table). A common approach involves providing multiple demonstrations to better disambiguate the desired task, but is cumbersome for the end user. Second, most current approaches for imitation learning require providing a new (set of) demonstration(s) for each desired task. Thus, if the user wishes to teach multiple tasks to the agent, the total number of demonstrations that need to be provided could be prohibitively large. Further, there are several ways of providing demonstrations to an agent, for instance, (1) kinesthetic teaching, wherein the human teacher holds and moves the robot's joints to have it complete the task, (2) teleoperation, in which the human teacher controls the robot's joints using controllers such as a joystick to have it complete the task, and (3) passive observation, wherein the human teacher performs the task which the agent observes. While kinesthetic teaching and teleoperation are more informative (as the learning agent gets an egocentric view of the task, and also receives information about the amount of force to apply at different steps, which is not possible to observe passively), they are also harder to provide, particularly for non-expert users.

In the second part of this dissertation (Chapters 5 and 6), we use natural language to reduce the number of demonstrations that the user needs to provide. Specifically, we propose a novel setting, wherein if the user has already provided a (set of) demonstration(s) of a task (the *source* task), and now wants to specify a related but different task (the *target* task), then instead of providing a new (set of) demonstration(s) for the target task, the difference between the source task and the target task is communicated using natural language.

Note that natural language can be used for task adaptation in any type of demonstration (kinesthetic, teleoperation, and passive). It is also worth noting that imitation learning is often used in conjunction with reinforcement learning, for instance, to initialize a policy from demonstrations that is then further improved using RL. Therefore, the methods in the first part of the dissertation can be combined with those in the second part, to build systems that leverage language for both RL and IL.

1.1.3 Other Related Problems

In addition to the motivations covered above, the techniques described in this dissertation also touch upon some related problems, which we discuss here.

Since language is a convenient and flexible modality, it can be used in a variety of ways to build intelligent agents, for instance, in (1) communicating the task, (2) providing feedback on the agent's performance, (3) guiding the agent to focus on the important aspects of the task, and (4) enabling the agent to ask clarification questions.

Using natural language for task specification, which is the focus of this dissertation, comes under communicating the task. However, the techniques in this dissertation are also related to other use cases of language listed above. For instance, techniques that use language

for task adaptation can also be used for feedback, where the agent's current (suboptimal) behavior can be seen as the "source demonstration", and the language specifies how the behavior should be modified to get the correct behavior. Similarly, the techniques we develop for generating auxiliary rewards from natural language task descriptions implicitly guide the agent to focus on the important aspects of the task. Thus, these different use cases are closely intertwined and progress along one is likely also beneficial for others.

Further, when humans demonstrate tasks to other humans, they often also use linguistic cues to augment the demonstration. For instance, when demonstrating a cooking recipe, a human might say, "Turn off the heat when the water starts boiling", as they demonstrate the action. Without the linguistic information, it could be difficult for the learner to infer when to take the action. Thus, using language and demonstrations jointly is a promising avenue to explore in the imitation learning setting. We will refer to this as *language-aided imitation learning* in the following.

We posit that the problem settings and techniques introduced in this dissertation can serve as useful building blocks for this future research direction. In particular, imitation learning often involves inferring a reward function from the demonstrations, followed by learning a policy using the inferred reward function. In the first part of this dissertation, we discuss approaches that use language in conjunction with an external reward function, and can therefore serve as a starting point for language-aided imitation learning, for instance, by inferring a reward function from the demonstrations, and then learning a policy that uses language with this reward function. Similarly, language-aided imitation learning requires combining complementary information from demonstrations and language. In the second part of this dissertation, we discuss approaches that use language for task adaptation, which

also involves combining information from a demonstration (of the source task) and a linguistic description (of the difference between the source and target tasks), and therefore, insights from language for task adaptation can be helpful for language-aided imitation learning.

1.2 Contributions and Dissertation Outline

Chapter 2 covers relevant background on sequential decision making, natural language processing, and language grounding, including a survey of the related work.

Chapters 3 through 6 cover the core technical contributions of this dissertation:

- In Chapter 3 [Goyal et al., 2019], we introduce a novel problem setting for RL, where in addition to the extrinsic reward function, the agent is also provided with a natural language description of the task. We develop a two-phase approach – LanguagE Action Reward Network (LEARN) – to use the task description for policy training: (1) the first phase involves learning a relatedness model between the action distribution in a trajectory and the task description using supervised learning; (2) the second phase involves training a policy for a new task given an extrinsic reward function and rewards generated from the task description using the relatedness model trained in phase one. We demonstrate that using language-based rewards in addition to the extrinsic rewards only leads to more sample-efficient learning and a better final policy, compared to using the extrinsic rewards only, on a diverse set of tasks in the Atari game Montezuma's Revenge.

- In Chapter 4 [Goyal et al., 2020b], we extend the previous approach to handle high-dimensional state sequences, so that the sequential information present in the states

can also be leveraged. We present modifications to the supervised learning phase and show that the new approach – Pixels and Language to Rewards (PixL2R) – leads to a more sample-efficient policy learning, both in sparse and dense reward settings, on a diverse set of continuous control robotic manipulation tasks.

- In Chapter 5 [Goyal et al., 2021], we introduce a novel problem setting for imitation learning, in which the agent needs to complete a target task, given the demonstration of a slightly different task (the source task) and a natural language description of the difference between the source and target tasks. We develop an approach – Language-Aided Reward and Value Adaptation (LARVA) – that decomposes the problem into two subproblems: (1) predicting the goal state for the target task given the source demonstration, and the natural language description of the difference between the source and target tasks, and (2) predicting the reward/value function for the target task, given the predicted goal state. Our experiments show that the approach successfully infers the target task more than 70% of the time, for different kinds of adaptations on a discrete multistep rearrangement domain.

- In Chapter 6 (under review), we propose alternate approaches for the setting introduced in the previous chapter. The first approach, relational reward adaptation, predicts the goal state for the target task given the goal state for the source task, and a distance metric between two states in the target task. The second approach, relational policy adaptation, assumes that the agent already has a policy for the source task, and learns a model to adapt the policy to the target task. We create two new benchmarks with adaptations that require reasoning about relationships between different entities

and show that both relational reward adaptation and relational policy adaptation are effective at learning policies for target tasks.

For all our experimental evaluations, we test our approaches on natural language collected from actual humans (as opposed to synthetic language generated using a grammar).

Finally, Chapter 7 lists several future directions that are worth exploring, and Chapter 8 includes concluding remarks that summarize the contributions of this dissertation.

Chapter 2

Background and Related Work

2.1 Sequential Decision Making

A large class of tasks in the real world can be formulated as an agent interacting with an environment, wherein at each step, the agent receives an observation from the environment and takes an action, and the environment transitions to a new state. The final state of the environment is, therefore, the result of a sequence of action decisions made by the agent. As such, these problems are referred to as *sequential decision making* problems. Examples include interacting with objects in the real world to accomplish a desired goal, such as cooking, rearranging a room, driving, or assembling furniture, as well as virtual environments, such as playing video games, or ordering an item from a website.

Many approaches have been proposed to train learning agents for such problems over the last several decades. Broadly, these approaches can be divided into two categories—reinforcement learning and imitation learning.

2.1.1 Terminology

In reinforcement learning (RL), the agent receives a reward from the environment at every timestep, which is the primary learning signal [Sutton and Barto, 2018]. The standard RL setup is often represented using the Markov Decision Process (MDP) formalism. An

MDP can be defined by the tuple $\langle S, A, T, R, \gamma \rangle$, where

- S is the **set of all states** in the environment,
- A is the **set of all actions** available to the agent,
- $T : S \times A \times S \to [0, 1]$ is the **transition function** that gives the probability of transitioning to state s' on taking action a in state s,
- $R : S \times A \to \mathbb{R}$ is the **reward function** which gives the numeric score for taking action a in state s, and
- $\gamma \in [0, 1]$ is the **discount factor** which downweights future rewards compared to immediate ones.

At timestep t, the agent observes the current state $s_t \in S$, and takes an action $a_t \in A$. The environment transitions to a new state $s_{t+1} \sim T(s_t, a_t, \cdot)$, and the agent receives a reward $R_t = R(s_t, a_t)$. The discounted sum of future rewards is called the **return**, $G_t = \sum_{i=t}^{T} \gamma^{i-t} R_i$.

The agent's behavior can be characterized by a **policy** $\pi : S \times A \to [0, 1]$ which gives the probability of taking action a in state s.

A **state-value function** $V^\pi : S \to \mathbb{R}$ is defined as the expected return from a state when following policy π. Given a policy π, the state-value function can be computed using the Bellman expectation equation,

$$V^\pi(s) = \sum_{a \in A} \pi(s, a) \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \right)$$

An **action-value function** $Q^\pi : S \times A \to \mathbb{R}$ is defined as the expected return when action a is taken at state s, and the policy π is followed thereafter. The action-value function for a policy π can be computed using an analogous Bellman expectation equation,

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \sum_{a' \in A} \pi(s',a') Q(s',a')$$

The objective in sequential decision making is to learn an **optimal policy** π^*, that achieves the maximum expected return. The optimal state- and action-value functions are denoted as V^* and Q^* respectively. In addition to the Bellman expectation equations, the optimal policy and the optimal value functions also satisfy the Bellman optimality equations:

$$V^*(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right)$$

and

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')$$

2.1.2 Reinforcement Learning

In RL, the objective is to learn an optimal policy that achieves the maximum expected return, given a reward function. Various approaches have been proposed to learn an optimal policy. Depending on whether they explicitly represent and learn the policy by maximizing the expected return, or learn the optimal value function and recover the optimal policy from that, these approaches can be classified as follows.

Policy-based methods. Policy-gradient methods learn an optimal policy π^* directly, by maximizing the expected total reward. One of the earliest approaches in this category is

REINFORCE [Williams, 1992], which uses Monte Carlo rollouts to estimate the gradient of the policy. Modifications of the basic algorithm have been proposed to reduce the variance of the gradient estimate.

Value-based methods. Value-based methods learn the optimal state-value function $V^*(s) = \max_\pi V^\pi(s)$ or the optimal action-value function $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, from which the optimal policy π^* can be recovered. Some of the earliest value-based methods include Q-learning [Watkins and Dayan, 1992] and SARSA [Rummery and Niranjan, 1994]. More recently, Mnih et al. [2015] proposed the deep Q-network (DQN), in which the action-value function is parameterized by a deep neural network, and learned using Q-learning. Several variations of the original DQN approach have since been proposed [Van Hasselt et al., 2016, Schaul et al., 2015, Wang et al., 2016, Bellemare et al., 2017, Fortunato et al., 2017, Hessel et al., 2018].

Actor-critic methods. Finally, actor-critic methods learn both a policy (the *actor*), and a value function (the *critic*), that are trained jointly. The critic is updated using the Bellman equation, while the actor is updated towards the direction suggested by the critic. Some popular actor-critic methods include trust-region policy optimization (TRPO; [Schulman et al., 2015]), proximal policy optimization (PPO; [Schulman et al., 2017]), deep deterministic policy gradient (DDPG; [Lillicrap et al., 2015]), and soft actor-critic (SAC; [Haarnoja et al., 2018]).

Instead of classifying RL algorithms based on whether they parameterize and learn the policy and/or the value function, we can also classify these approaches based on whether

they use the current policy to gather data for updating the parameters, or use data from elsewhere.

On-policy algorithms. These algorithms use the current policy to gather data for updating the parameters. Examples of on-policy algorithms include REINFORCE, TRPO, PPO, and SARSA.

Off-policy algorithms. These algorithms optimize for the policy or the value function using data that may not be generated by the current policy. For policy-based methods, this involves maintaining two policies – the target policy, which is being updated towards the optimal policy, and the behavior policy, which is used to collect the data. DDPG is an example of such an approach, where the behavior policy is a noisy version of the target policy. For value-based methods, since there is no explicit policy, data is collected using the current value function, but stored in an experience replay buffer. To update the value function, data is sampled from the experience replay buffer; hence, the data used at a particular update step may have been collected from an older snapshot of the value function. Examples of such approaches include Q-learning, DQN, DDPG, and SAC.

Off-policy algorithms are generally more sample efficient compared to on-policy algorithms, but are known to be less stable [Sutton and Barto, 2018].

In this dissertation, we use the PPO algorithm for our RL experiments, since it supports both discrete and continuous domains, and was found to be reasonably stable and sample-efficient on the domains used. However, the techniques we develop are agnostic to the choice of the RL algorithm.

While reinforcement learning has been successfully applied in multiple domains, several challenges remain that need to be addressed to make RL practical for real-world applications.

First, specifying a reward function that accurately and efficiently encodes the desired task may be non-trivial. A common approach that works for simple tasks consists of using a sparse reward—for instance, the agent receives a non-zero reward on completing the task, and a zero reward in all other cases. To learn from such a reward function, the agent must explore without much feedback from the environment, which could become prohibitively time-consuming as the complexity of tasks grows. One way to address this limitation is to define a dense reward function, which is non-zero at most timesteps, giving the agent additional signal as it is making progress towards the goal [Ng et al., 1999]. However, designing dense rewards is itself challenging, and may lead to reward hacking, where the agent finds an undesired behavior that achieves a high reward [Amodei and Clark, 2016].

Another challenge in RL is sample efficiency—even for hand-designed reward functions, the agent might need a considerable number of interactions with the environment to learn the desired behavior. Approaches to address this involve using expert data [Brys et al., 2015], curiosity-driven exploration in which the agent is rewarded for visiting new regions of the state space [Burda et al., 2018, Achiam and Sastry, 2017, Schmidhuber, 1991], hierarchical RL in which a low-level policy is trained to reach different subgoals and a high-level policy is trained to predict the next subgoal [Barto and Mahadevan, 2003, Vezhnevets et al., 2017], and transfer learning in which a policy trained for one task is adapted for a different task [Taylor and Stone, 2009].

Finally, when policies are trained or deployed in the real world, there are safety

concerns that must be taken into account [Garcıa and Fernández, 2015]. Common approaches to safe RL include adding constraints to prevent the agent from visiting unsafe states, and modifying the reward function such that unsafe states have a large negative value.

In this dissertation, we present approaches that use natural language to address the challenges of reward design and sample efficiency (Chapters 3 and 4). We show that rewards can be generated from natural language descriptions, which addresses the reward design problem, particularly for non-experts. Further, the rewards generated from natural language help improve the sample efficiency of the learner, both in sparse and dense reward settings.

2.1.3 Imitation Learning

The imitation learning problem can be described using an MDP\R $= \langle S, A, T, \gamma \rangle$, where the symbols denote the state space, the action space, the transition function, and the discount factor, as detailed in Section 2.1.2. Instead of the reward, the agent has access to a set of expert demonstrations $\tau_1, \ldots, \tau_N$. The goal of imitation learning is to infer the demonstrator's intent, and thereby learn a policy $\pi : S \times A \rightarrow [0, 1]$ that maximizes the return, under the unknown expert reward function.

Approaches in imitation learning can be classified into the following categories.

Behavior cloning. Given a set of demonstrations $\tau_1, \ldots, \tau_N$, where $\tau_i = ((s_{ij}, a_{ij}))_{j=1}^{N_i}$, a policy $\pi : S \times A \rightarrow [0, 1]$ is learned using the state-action pairs in all the demonstrations, in a supervised learning setting [Pomerleau, 1991]. When used for predictions on unseen states, minor errors in prediction cause the agent to diverge from the data distribution the policy was trained on, leading to compounding errors. Several methods have been proposed that

address this limitation [Ross et al., 2011, Venkatraman et al., 2015, Brantley et al., 2019].

Inverse Reinforcement Learning. In these approaches, a reward function is inferred from the demonstrations, and a policy is learned using RL on the recovered reward function [Ng et al., 2000, Ziebart et al., 2008, Ramachandran and Amir, 2007].

Adversarial Imitation Learning. These approaches learn a policy jointly with a discriminator, such that the discriminator tries to distinguish states visited by the policy from the states visited by the expert, and the policy tries to visit states so as to confuse the discriminator [Ho and Ermon, 2016, Torabi et al., 2018].

Imitation learning suffers from some of the same challenges as RL. We elaborate some of these, as well as describe some additional challenges specific to imitation learning.

Imitation learning is fundamentally an ill-defined problem—given a set of demonstrations, there are multiple reward functions under which the demonstrations are optimal. However, not all these reward functions lead to policies that may be desirable under the demonstrator's true reward function. To address this, several approaches have been proposed that model this uncertainty [Ziebart et al., 2008, Ramachandran and Amir, 2007], obtain feedback/corrections from a human [Cui and Niekum, 2018, Niekum et al., 2013], or formulate the problem differently such as learning from pairwise rankings between a set of demonstrations [Brown et al., 2019].

Another major challenge in imitation learning is that of sample efficiency—providing demonstrations is often cumbersome, and therefore, we would like the agent to learn from as few demonstrations as possible. Recently, several approaches have been proposed that use

meta-learning [Thrun and Pratt, 2012], to enable few-shot imitation learning [Finn et al., 2017, Duan et al., 2017, Pathak et al., 2018].

In this dissertation, we take a step towards addressing sample efficiency in imitation learning, using language. We propose a framework which enables reusing demonstrations from related tasks, where the difference between the target task and the demonstrated task is specified using language (Chapters 5 and 6). This allows the agent to learn a new task in a zero-shot setting, that is, without any new demonstrations.

Transfer Learning. This setting is related to the problem of transfer learning in artificial intelligence, where data from a source task is used to learn a target task more efficiently [Bozinovski and Fulgosi, 1976, Weiss et al., 2016]. Of note here are transfer learning approaches that are designed for reinforcement learning [Taylor and Stone, 2009, Zhu et al., 2020]. These approaches can be divided into categories based on the difference between the source and target tasks (e.g. change in the state space, action space, initial state distribution, transition function, or reward function), whether the single source task is selected by a human or if the agent needs to pick (a subset of) source task(s) from a set of source tasks most relevant to the target task, and the type of knowledge transferred (e.g. value function, policy, features). In our work, the source and target tasks differ in the goal state (i.e., the reward function), and the experiments we run assume identical state space, action space and transition function for the source and target tasks. However, since a policy is learned for the target task from an inferred reward function using RL, differences in the transition function should be relatively straightforward to handle using the approaches we present.

Relational Reasoning. The approach we present for the setting above in Chapter 6 is based on relational reasoning, which involves representing the inputs of a learning system using a set of *entities*, and the model learns to map the inputs to the desired outputs by reasoning about the relationships between these entities. Various techniques have been proposed for relational reasoning [Scarselli et al., 2008, Kipf and Welling, 2016, Velickovic et al., 2017, Goyal et al., 2020a]. These approaches have been shown to be effective for various machine learning problems, including reinforcement learning [Džeroski et al., 2001, Zambaldi et al., 2018, Zhou et al., 2022], language grounding [Santoro et al., 2017, Dong et al., 2020], and modeling passive dynamics [Didolkar et al., 2021]. While relational RL represents the policy using a relational model which reasons about the state represented in terms of entities, the approach we present uses a relational model to infer a reward function or a policy initialization for the target task. As such, our approach is orthogonal to relational RL methods, and can be combined with them easily. Another line of work involves using language to describe entity dynamics in different environments, which results in efficient policy learning on new environments [Narasimhan et al., 2018]. This is also orthogonal to our approach, as we use language to specify change in goals, instead of specifying entity dynamics.

2.2 Natural Language Processing

Natural languages such as English, Mandarin, Hindi, Spanish, and French evolved with humans over hundreds of thousands of years, and form the basis of the human civilization. Unlike programming languages, which are carefully designed by programmers to unambiguously communicate programmers' intent to a computer, natural languages are often

ambiguous.

Natural language processing (NLP) refers to the subfield of AI that deals with building intelligent systems that can communicate with humans using natural language, instead of programs. This is an important requirement if we are to have intelligent systems working alongside humans, since most humans are not computer programmers, and will want to communicate with these systems using natural language, much like how they communicate with other humans. NLP can be divided into two main categories—natural language understanding, and natural language generation.

Problems in natural language understanding include disambiguating words with multiple meanings using the context they appear in (e.g. inside the *building* vs. *building* a fence), linking pronouns to the corresponding nouns, and linking entities mentioned in the text to real-world entities. Problems in natural language generation include generating grammatically correct text, and generating a sequence of sentences that form a coherent narrative.

In this dissertation, we only deal with natural language understanding, since the problem settings we consider involve communicating a task to an agent using natural language. We briefly review some natural language understanding approaches below.

Some of the earliest approaches in natural language understanding used rule-based methods [Winograd, 1971, Shank and Abelson, 1977]. This was followed by the development of statistical methods for tasks such as part-of-speech tagging [Church, 1989], machine translation [Brown et al., 1990], parsing [Magerman, 1995, Collins, 1996, Charniak, 1997], and information extraction. Statistical approaches continued to be developed, with new models being proposed, such as conditional random fields [Lafferty et al., 2001] and latent Dirichlet

allocation [Blei et al., 2003].

More recently, neural network based approaches became popular in NLP. Collobert et al. [2011] presented neural methods for standard NLP tasks, such as part-of-speech tagging, chunking, named entity recognition, and semantic role labeling. Sutskever et al. [2014], Bahdanau et al. [2014] developed neural approaches for sequence-to-sequence tasks.

Of note for this dissertation are methods that encode words and sentences into dense vector representations. We briefly review some of the approaches below are relevant for the approaches we present in later chapters.

GloVe Embeddings. Pennington et al. [2014] proposed learning vector representations of words – Global Vectors (GloVe) – such that the dot product of the vector representations of words i and j, $w_i^T w_j$ is close to the co-occurrence probability of words i and j in the training corpus. This results in vector representations of words that capture useful semantic information, as words that appear in similar contexts get similar representations. These vectors can be pretrained on large text corpora, and then used for downstream NLP tasks. However, a key limitation of this approach (and other related approaches that learn word embeddings, like Mikolov et al. [2013]) is that they encode words into vectors independent of the context. As such, words that have multiple meanings (e.g. the word "bank" may refer to a financial institution or the edge of a river) will have a single vector representation for both these meanings under this model, which is not ideal.

InferSent. Building on the idea of word embeddings, Conneau et al. [2017] trained a model that can encode arbitrary sentences into dense vector representations. The model encodes

each word in the input sentence using GloVe vectors, which is then passed through a bidirectional LSTM, followed by a maxpooling operation to get the final vector representation of the sentence. Using a pretrained sentence encoder may be better than using word embeddings for downstream tasks that may not have enough data to learn the sentence encoder from scratch.

BERT embeddings. As the transformer model based on self-attention [Vaswani et al., 2017] rose to popularity, the transformer-based Bidirectional Encoder Representations from Transformers (BERT) model [Devlin et al., 2018] became the widely used technique for sentence encoding. The input sentence is passed through a sequence of transformer layers to output contextualized representations of each token. Unlike RNN-based models that process the input tokens sequentially, transformer-based models compute attention scores between all pairs of input tokens. This allows them to capture long-range dependencies between input tokens, and consequently outperform RNN-based approaches.

CLIP embeddings. Instead of training a text encoding model purely from textual data as in BERT, the Contrastive Language-Image Pre-training (CLIP) approach Radford et al. [2021] trains a transformer-based model from paired image and language data, which can be used for downstream multimodal tasks such as image-to-text or text-to-image retrieval. Further, because these models are trained on multimodal data, the text embeddings contain visual information, and vice versa, which can be useful for many text-only or image-only tasks as well.

2.3 Language Grounding

Language grounding, or symbol grounding, refers to the problem of mapping linguistic concepts to the agent's perception and actions [Harnad, 1990]. For example, to understand an instruction such as "Place the mug next to the book", a learning agent needs to map *mug* and *book* to objects in the environment, *place* to an action, and *next to* to a spatial relation in the environment.

2.3.1 Grounding Language to Images and Videos

In the last few years, many problems have been introduced that require grounding linguistic concepts to images or videos, such as image captioning [You et al., 2016, Anderson et al., 2018a], video captioning [Venugopalan et al., 2015], visual question-answering [Antol et al., 2015], and identifying regions of an image referred to by a natural language expression [Kazemzadeh et al., 2014].

2.3.2 Instruction-following

In a large class of problems, which can broadly be termed as *instruction-following*, language is used to communicate the task to a learning agent. In this setting, the agent is given a natural language command for a task, and is trained to take a sequence of actions that completes the task. A popular approach for instruction-following involves using graphical models that are instantiated based on the structure of the natural language command, from which the most likely plan is inferred [Tellex et al., 2011, Howard et al., 2014b,a]. Other approaches involve parsing the input command into a rich grammar formalism [Artzi and Zettlemoyer, 2013], using specialized neural network architectures to map instructions to

visual goals [Misra et al., 2018] position visitation [Blukis et al., 2018b, 2019], intermediate goals [Paxton et al., 2019], or actions [Bisk et al., 2016, Stepputtis et al., 2020].

A well-studied problem in this setting is Vision-and-language Navigation, where the tasks consist of navigating to a desired location in an environment, given a natural language command and an egocentric view from the agent's current position [Anderson et al., 2018b, Fried et al., 2018, Wang et al., 2019]. Another subcategory of instruction-following involves instructing an embodied agent using natural language [Tellex et al., 2011, Hemachandra et al., 2015, Arkin et al., 2017, Shridhar et al., 2020, Stepputtis et al., 2020, Misra et al., 2016, Sung et al., 2018, Blukis et al., 2018a, 2019, Shao et al., 2020].

The approaches that we present are different from instruction-following in that we use language as an *auxiliary signal* in addition to the main supervisory signal for the setting (i.e. reward for RL and demonstrations for IL). In particular, problems that require fine-grained low-level control with complex transition dynamics, such as a robot manipulation task, are usually challenging for instruction-following approaches without an explicit reward function. Since our approaches use (or infer from demonstrations) an explicit reward function, RL can be used to learn a policy even in domains with complex transition dynamics. An alternate way to think about the distinction between instruction-following and using language to infer rewards is that language may often communicate *what to do*, rather than *how to do a task*. As such, mapping instructions to actions directly, as in most instruction-following approaches, may not be always ideal, and it would be better to infer a reward function from language that captures the task objectives, which can then be used to learn a policy.

2.3.3 Language to Aid Learning

A number of approaches have been proposed that use language to aid a learning agent.

Some of these approaches use language to generate plans. Branavan et al. [2012a] extract preconditions from text to generate plans for games. Sung et al. [2018] use language in conjunction with trajectories and point clouds to plan manipulation trajectories to interact with unseen objects. Nyga et al. [2018] use language in a dialog setting to fill incomplete plans. These approaches are different from our framework presented in Chapters 3 and 4, in that we incorporate language into reinforcement learning, instead of planning.

Among the approaches that use language in an RL setting, Narasimhan et al. [2017, 2018] use language to transfer concepts from one task to another to speed up RL, Kuhlmann et al. [2004] use language to modify the Q-function of an RL agent, Branavan et al. [2012b] use language to generate additional features for speeding up learning, Kaplan et al. [2017] use a predefined set of natural language commands which are used to generate additional rewards to train an RL agent, Hutsebaut-Buysse et al. [2020b] use pretrained word embeddings to make goal-conditioned RL sample efficient, Abolghasemi et al. [2018] use natural language to attend to specific parts of the state to learn more robust visuomotor policies, Wang and Narasimhan [2021] learn to ground language to entities and dynamics, simultaneously with the policy, and Tambwekar et al. [2021] propose initializing decision-tree based policies using natural language.

Language has also been used to incorporate human feedback. For instance, Broad et al. [2017] propose using natural language corrections for robotic manipulators, in which a

trajectory may be paused and corrected by a human signaling the agent to replan the trajectory. Co-Reyes et al. [2018] propose iterative natural language corrections to communicate the goal to a learning agent. Mehta and Goldwasser [2019] propose using natural language for advice, in an instruction-following setting. Since these methods do not involve generating reward signals from natural language, they are orthogonal to the approaches presented in this dissertation, and can be combined with them to learn a better policy using RL. Sumers et al. [2020] use linguistic feedback by mapping language to features and sentiment, and training a linear reward model that takes in the features and outputs the sentiment score. Thus, the sentiment scores are treated as the ground truth reward values. However, these rewards are a function of only the language description (which is provided for the last executed trajectory in a human-in-the-loop setting), whereas the approaches we present infer rewards as functions of both language and trajectories. As such, we can combine these two approaches—first use the approaches we develop to learn a general reward function for the entire task, and then use the approach above to get additional rewards per trajectory from humans interactively.

Andreas et al. [2017] propose to exploit compositionality in natural language to aid generalization, by mapping tasks into a latent space of language. Hutsebaut-Buysse et al. [2020a] propose a transfer learning framework, wherein a new task can be learned from a set of previously learned tasks, each of which is described using language. While related to our setting introduced in Chapters 5 and 6, in these methods, language is provided for each task independently, and tasks are deemed similar if their linguistic descriptions are related. In our setting, however, language is used to explicitly describe the difference between two tasks. Kamlish et al. [2019] use natural language commentary in the domain of chess to learn

an evaluation function for moves, but is different from our approaches, as in this approach, language is only used at training time whereas for the frameworks we propose, a key element is using language at test time to aid learning a *new* task not seen during training.

Chapter 3

Using Natural Language for Reward Shaping in Reinforcement Learning

3.1 Introduction

One of the key challenges in applying reinforcement learning to a problem is designing reward functions that accurately and efficiently describe tasks. For the sake of simplicity, a common strategy is to provide the agent with sparse rewards. A reward function is said to be sparse if the distribution of reward values over all the states in the environment has entropy close to zero. For example, in a goal-based task, the agent may be given a positive reward upon reaching the goal state, and zero reward otherwise. However, it is well-known that learning is often difficult and slow in sparse reward settings [Večerík et al., 2017]. By contrast, dense reward functions are those for which the distribution of reward values over all the states in the environment has a high entropy. Such rewards can be easier to learn from, but are significantly more difficult to specify. In this work, we address this issue by using natural language to provide dense rewards to RL agents in a manner that is easy to specify.

Consider the scenario in Figure 3.1 from the Atari game Montezuma's Revenge. Suppose we want the agent to go to the left while jumping over the skull (as shown by the blue trajectory). If the agent is given a positive reward only when it reaches the end of the desired

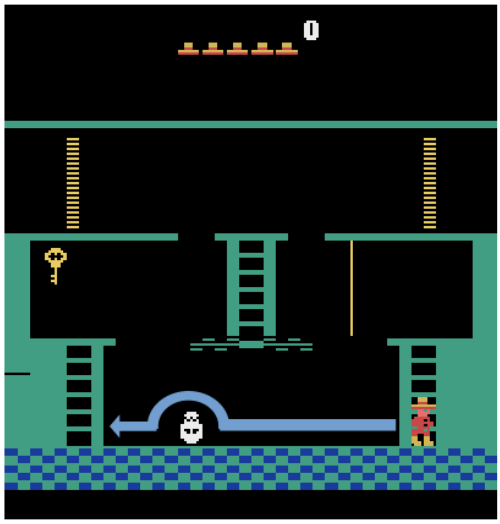


Figure 3.1: An agent exploring randomly to complete the task described by the blue trajectory may need considerable amount of time to learn the behavior. By giving natural language instructions like "Jump over the skull while going to the left", we can give intermediate signals to the agent for faster learning.

trajectory, it may need to spend a significant amount of time exploring the environment to learn that behavior. Giving the agent intermediate rewards for progress towards the goal can help, a technique known as "reward shaping" [Ng et al., 1999]. However, designing intermediate rewards is hard, particularly for non-experts.

Instead, we propose giving the agent intermediate rewards using instructions in natural language. For instance, the agent can be given the following instruction: "Jump over the skull while going to the left" to provide intermediate rewards that accelerate learning. Since natural language instructions can easily be provided even by non-experts, it will enable them to teach RL agents new skills more conveniently.

The main contribution of this work is a new framework which takes an arbitrary natural language instruction and the trajectory executed by the agent so far, and makes

a prediction whether the agent is following the instruction, which can then be used as an intermediate reward.

Using arbitrary natural language statements within reinforcement learning presents several challenges. First, a mapping between language and objects/actions must implicitly or explicitly be learned, a problem known as *symbol grounding* [Harnad, 1990]. For example, to make use of the instruction, "Jump over the snake", the system must be able to ground "snake" to appropriate pixels in the current state (assuming the state is represented as an image) and "jump" to the appropriate action in the action space. Second, natural language instructions are often incomplete. For instance, it is possible that the agent is not directly next to the snake and must walk towards it before jumping. Third, natural language inherently involves ambiguity and variation. This could be due to different ways of referring to the objects/actions (e.g. "jump" vs. "hop"), different amounts of information in the instructions (e.g. "Jump over the snake" vs. "Climb down the ladder after jumping over the snake"), or the level of abstraction at which the instructions are given (e.g. a high-level subgoal: "Collect the key" vs. low-level instructions: "Jump over the obstacle. Climb up the ladder and jump to collect the key.")

Once an instruction has been interpreted, we incorporate it into the RL system as an additional reward (as opposed to other alternatives like defining a distribution over actions), since modifying the reward function allows using any standard RL algorithm for policy optimization. We evaluate our approach on Montezuma's Revenge, a challenging game in the Atari domain [Bellemare et al., 2013], demonstrating that it effectively uses linguistic instructions to significantly speed learning, while also being robust to variation in instructions.

3.2 Approach

In this work, we consider an extension of the MDP framework, defined by $\langle S, A, R, T, \gamma, l \rangle$, where $l \in L$ is a language command describing the intended behavior (with L defined as the set of all possible language commands). We denote this language-augmented MDP as MDP+L. Given an MDP(+L), reinforcement learning can be used to learn an optimal policy $\pi^* : S \rightarrow A$ that maximizes the expected sum of rewards. We use R_{ext} ("extrinsic") to denote the MDP reward function above, to avoid confusion with language-based rewards that we define in Section 3.2.2.

In order to find an optimal policy in an MDP+L, we use a two-phase approach (Figure 3.2):

LanguagE-Action Reward Network (LEARN). In this step, we train a neural network that takes paired (trajectory, language) data from the environment and predicts if the language describes the actions within the trajectory. To train the network, we collect natural language instructions for trajectories in the environment (Section 3.2.1).

Language-aided RL. This step involves using RL to learn a policy for the given MDP+L. Given the trajectory executed by the agent so far and the language instruction, we use LEARN to predict a relatedness score between the executed trajectory and the instruction, which can be considered as a measure of progress towards the goal. As such, the relatedness score is used to define a shaping reward (Section 3.2.2). Note that since we are only modifying the reward function, this step is agnostic to the particular choice of RL algorithm.

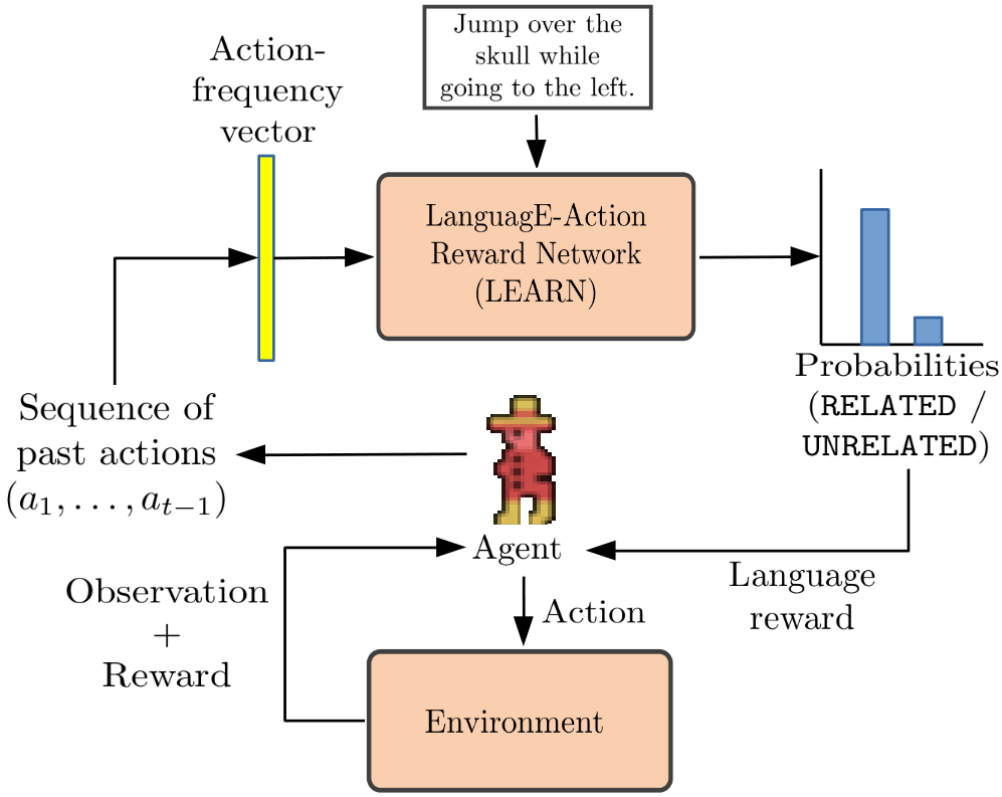


Figure 3.2: Our framework consists of the standard RL module containing the agent-environment loop, augmented with a LanguagE-Action Reward Network (LEARN) module.

3.2.1 LanguagE-Action Reward Network

LEARN takes in a trajectory and a language description and predicts whether the language describes the actions in the trajectory. More formally, given a trajectory τ, we create *action-frequency vectors* from it as follows:

1. Sample two distinct timesteps i and j (such that $i < j$) from the set $\{1, \ldots, |\tau|\}$, where $|\tau|$ denotes the number of timesteps in τ. Let $\tau[i:j]$ denote the segment of τ between timesteps i and j.

2. Create an *action-frequency vector* f from the actions in $\tau[i:j]$, where the dimensionality of f is equal to the number of actions in the MDP+L, and the k^{th} component of f is the fraction of timesteps action k appears in $\tau[i:j]$.

Using the above process, we create a dataset of (f, l) pairs from a given set of (τ, l) pairs. Positive examples are created by sampling f from a given trajectory τ and using the language description l associated with τ. Negative examples are created by (1) sampling an action-frequency vector f from a given trajectory τ, but choosing an alternate language description l' sampled uniformly at random from the data excluding l, or (2) creating a random action-frequency vector f' and pairing it with the language description l. These examples are used to train a neural network, as described below. Thus, given a pair (f, l), the network learns to predict whether the action-frequency vector f is related to the language description l or not.

Next, we describe the details of the neural network. The action-frequency vector is passed through a sequence of fully-connected layers to get an encoded action vector with dimension D_1. To embed the natural language instruction into a D_2-dimensional vector, we

experimented with three models:

1. **InferSent** : Here, we use a pretrained sentence embedding model [Conneau et al., 2017], which embeds sentences into a 4096-dimensional vector space. The 4096-dimensional vectors were projected to D_2-dimensional vectors using a fully-connected layer. We train only the projection layer during training, keeping the original sentence embedding model fixed.

2. **GloVe+RNN** : In this model, we represent the sentence using pretrained 50-dimensional GloVe word embeddings [Pennington et al., 2014], and train a two-layer GRU [Cho et al., 2014] encoder on top of it, while keeping the word embeddings fixed. We used the mean of the output vectors from the top layer as the encoding of the sentence. The hidden state size of the GRUs was set to D_2.

3. **RNNOnly** : This model is identical to GloVe+RNN, except instead of starting with pretrained GloVe vectors, we randomly initialize the word vectors and train both the word embeddings and the two-layer GRU encoder.

These three models trade-off prior domain knowledge with flexibility – InferSent model starts with the knowledge of sentence similarity and is the least flexible, GloVe+RNN model starts with word vectors and is more flexible in combining them to generate sentence embeddings, while RNNOnly starts with no linguistic knowledge and is completely flexible while learning word and sentence representations. More details about these models are discussed in Section 2.2.

The encoded action-frequency vector and language vector are then concatenated, and further passed through another sequence of fully-connected layers, each of dimension D_3,

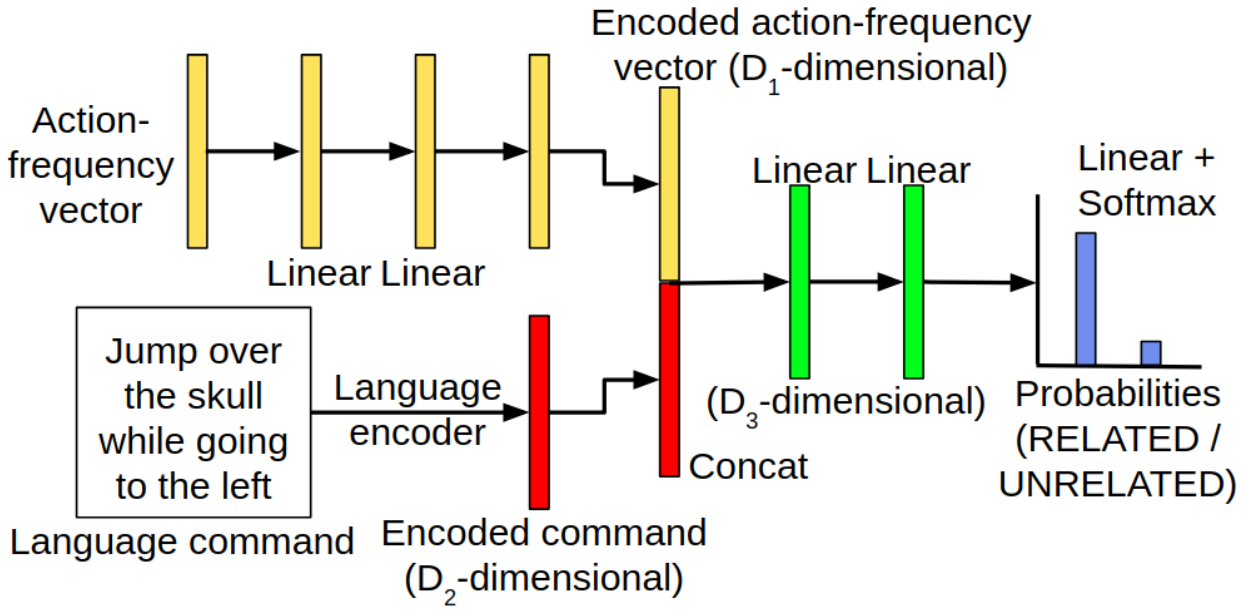


Figure 3.3: Neural network architecture for LEARN (Section 3.2.1)

followed by a softmax layer. The final output of the network is a probability distribution over two classes – `RELATED` and `UNRELATED`, corresponding to whether the action-frequency vector f can be explained by the language instruction l. Our complete neural network architecture is shown in Figure 3.3. D_1, D_2 and D_3 were tuned using validation data.

We used backpropagation with an Adam optimizer (Kingma and Ba [2014]) to train the above neural network for 50 epochs to minimize cross-entropy loss.

3.2.2 Using Language-based Rewards in RL

To incorporate language information into RL, we use LEARN's predictions to generate intermediate rewards. Given the sequence of actions $a_1, \ldots, a_{t-1}$ executed by the agent until timestep t and the language instruction l associated with the given MDP+L, we create

an action-frequency vector f_t, by setting the k^{th} component of f equal to the fraction of timesteps action k appears in $a_1, \ldots, a_{t-1}$. The resulting action-frequency vector f and the language instruction l are passed to LEARN. Let the output probabilities corresponding to classes `RELATED` and `UNRELATED` be denoted as $p_R(f_t)$ and $p_U(f_t)$. Note that since l is fixed for a given MDP+L, $p_R(f_t)$ and $p_U(f_t)$ are functions of only the current action-frequency vector f_t.

Intuitively, trajectories that contain actions described by the language instruction more often will have higher values of $p_R(f_t)$, compared to other trajectories. For instance, if the language instruction is "Jump over the skull while going to the left", then trajectories with high frequencies corresponding to the "jump" and "left" actions will be considered more related to the language by LEARN. Therefore, we can use these probabilities to define intermediate language-based rewards. These intermediate rewards will enable the agent to explore more systematically, by choosing relevant actions more often than irrelevant actions.

To map the probabilities to language-based shaping rewards, we define a potential function for the current timestep as $\phi(f_t) = p_R(f_t) - p_U(f_t)$. The intermediate language-based reward is then defined as $R_{lang}(f_t) = \gamma \cdot \phi(f_t) - \phi(f_{t-1})$, where γ is the discount factor for the MDP+L.

Ng et al. [1999] showed that a shaping reward of the form $R_{sh}(s_t) = \gamma \cdot \phi(s_t) - \phi(s_{t-1})$, for any potential function ϕ of the state does not change the optimal policy. Since the potential function we use in our approach depend on the action-frequency vector f_t, we need to ensure that the modified reward function does not change the optimal policy, which we show below.

Theorem. *Let $M = \langle S, A, T, R_{ext}, \gamma \rangle$ be a given MDP with initial state distribution ρ_0, and $R_{lang}(f_t, t) = \gamma \cdot \phi(f_t, t) - \phi(f_{t-1}, t)$ be a shaping reward function, where f_t is the action-frequency vector corresponding to actions $a_1, \ldots, a_t$ as defined in Section 3.2.1, t is the current timestep, and ϕ is a function such that $\phi(\cdot, t) = 0$ for the last timestep of any trajectory. Then, an optimal policy in M is also an optimal policy for the modified reward function $R_{ext} + R_{lang}$.*

Proof. The optimal policy under the original reward function is also an optimal policy under the modified reward function, if

$$\forall \pi, \qquad \mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}}(s_0)] = \mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}+R_{lang}}(s_0)] + C$$

where C is a term independent of π.

Defining ρ_t to be the state distribution induced by π after t timesteps, we get

$$\begin{aligned}
&\mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}+R_{lang}}(s_0)] \\
&= \mathbb{E}_{s_0 \sim \rho_0}[\sum_{t=0}^{T} \gamma^t (R_{ext}(s_t) + \gamma \cdot \phi(f_t, t) - \phi(f_{t-1}, t))] \\
&= \mathbb{E}_{s \sim \rho_0}[\sum_{t=0}^{T} \gamma^t R_{ext}(s_t)] + \mathbb{E}_{s_0 \sim \rho_0}[\sum_{t=0}^{T} \gamma^t (\gamma \cdot \phi(f_t, t) - \phi(f_{t-1}, t))] \\
&= \mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}}(s_0)] + \mathbb{E}_{s_0 \sim \rho_0}[\sum_{t=0}^{T} \gamma^t (\gamma \cdot \phi(f_t, t) - \phi(f_{t-1}, t))] \\
&= \mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}}(s_0)] + \sum_{t=0}^{T} \left(\gamma^{t+1} \mathbb{E}_{s \sim \rho_t}[\phi(f_t, t)] - \gamma^t \mathbb{E}_{s \sim \rho_{t-1}}[\phi(f_t, t)]\right) \\
&= \mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}}(s_0)] + \gamma^T [\phi(f_T, T)] - \mathbb{E}_{s_0 \sim \rho_0}[\phi(f_0, 0)] \\
&= \mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}}(s_0)] - \mathbb{E}_{s_0 \sim \rho_0}[\phi(f_0, 0)]
\end{aligned} \tag{3.1}$$

where we use $\phi(f_T, T) = 0$ in the last step. Further, $\mathbb{E}_{s_0 \sim \rho_0}[\phi(f_0, 0)]$ is a term that depends on the initial action-frequency vector containing all zero entries, and is therefore independent

of the policy π. Hence, $\mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}+R_{lang}}(s_0)]$ and $\mathbb{E}_{s_0 \sim \rho_0}[V^{\pi}_{R_{ext}}(s_0)]$ are identical modulo a term independent of the policy.

Thus, an optimal policy in M is also an optimal policy for the modified reward function $R_{ext} + R_{lang}$.

□

3.3 Domain and Dataset

To validate the effectiveness of our approach, we conducted experiments on the Atari game Montezuma's Revenge. The game involves controlling an agent to navigate around multiple rooms. There are several types of objects within the rooms – (1) ladders, ropes, doors, etc. that can be used to navigate within a room, (2) enemy objects (such as skulls and crabs) that the agent needs to escape from, (3) keys, daggers, etc. that can be collected. A screenshot from the game is included in Figure 3.1. We selected this game because the rich set of objects and interactions allows for a wide variety of natural language descriptions.

To collect data for training LEARN, we generate trajectories in the environment, which may or may not be directly relevant for the final task(s). Then, for each trajectory, we get natural language annotations from human annotators, which are in the form of instructions that the agent should follow to go from the initial state of the trajectory to the final state.

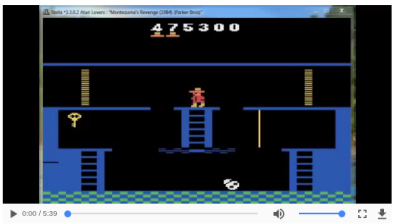
In our experiments, we used 20 trajectories from the Atari Grand Challenge dataset [Kurin et al., 2017], which contains hundreds of crowd-sourced trajectories of human game-plays on 5 Atari games, including Montezuma's Revenge. The 20 trajectories we used contain

a total of about 183,000 frames. From these trajectories, we extracted 2,708 equally-spaced clips (with overlapping frames), each three-seconds long. To obtain language descriptions for these clips, we used Amazon Mechanical Turk. Workers were shown clips from the game and asked to provide corresponding language instructions. Each annotator was asked to provide descriptions for 6 distinct clips, while each clip was annotated by 3 people. Figure 3.4 shows the interface used on Amazon Mechanical Turk for data collection.

To filter out bad annotations, we manually looked at each set of 6 annotations and discarded the set if any of them were generic statements (e.g. "Good game!", "Well played."), or if all the descriptions were very similar to one another (therefore suggesting that they are likely not related to the corresponding clips). After minimal filtering, we obtained a total of 6,870 language descriptions. Note that the resulting dataset may still be quite noisy, since our filtering process doesn't explicitly check if the language instructions are related to the corresponding clips, nor do we correct for any spelling or grammatical errors.

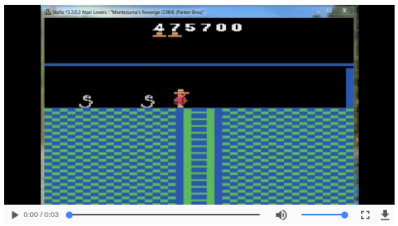
Table 3.1 shows 20 randomly selected annotations after filtering. It can be observed that the annotations have a significant amount of variation, both in terms of length and vocabulary. Further, several descriptions (1) contain spelling errors (e.g. "climbling" in annotation 6 and "dwon" in annotation 7), (2) are ill-formed (e.g. annotation 2) or (3) are not very informative (e.g. annotations 1 and 7). We do not filter out or correct these annotations, as the process requires significant manual effort. Thus, our method is able to extract useful information from these annotations even in the presence of noise.

First, watch the first 1-2 minutes of the video below of a game play:



Now, you will be shown clips from the above game. Assuming your friend is playing the game, what would you tell them so that they play the game as shown in the clip? Look at the examples below to better understand the task.

Example 1: Watch the clip below:



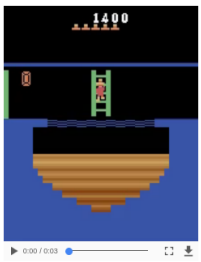
One possible description for this clip could be the following:
Go slightly to the right and climb down the ladder.

Example 2: Watch the clip below:

One possible description for this clip could be the following:
Jump once while going left.

Finally, as shown in the above examples, write one description for each of the clips below:

Clip 1:



Please enter the description below:

Figure 3.4: Sample Mechanical Turk task for collecting natural language descriptions.

1.	wait
2.	using the ladder on standing
3.	going slow and climb down the ladder
4.	move down the ladder and walk left
5.	go left watch the trap and move on
6.	climbling down the ladder
7.	ladder dwon and running this away
8.	stay in place on the ladder.
9.	go down the ladder
10.	go right and climb up the ladder
11.	just jump and little move to right side
12.	run all the way to the left.
13.	go left jumping once
14.	go left
15.	move right and jump over green creature then go down the ladder
16.	hop over to the middle ledge
17.	wait for the two skulls and dodge them in the middle
18.	walk to the left and then jump down
19.	jump to collected gold coin and little move
20.	wait for the platform to materialize then walk and leap to your right to collect the coins.

Table 3.1: Examples of descriptions collected using Amazon Mechanical Turk

3.4 Experimental Evaluation

The (trajectory, language) pairs collected using the process described in Section 3.3 were split into training and validation sets, such that there is no overlap between the frames in the training set and the validation set. More specifically, Level 1 of Montezuma's revenge consists of 24 rooms, of which we use 14 for training, and the remaining 10 for validation and testing. The set of objects in both training and validation/test splits are the same, but each room has only a subset of these objects arranged in different layouts. We create

a training dataset with 160,000 (action-frequency vector, language) pairs from the training set, and a validation dataset with 40,000 pairs from the validation set, which were used to train LEARN.

For reinforcement learning, we define a set of 15 diverse tasks in multiple rooms, each of which requires the agent to go from a fixed start position to a fixed goal position while interacting with some of the objects present in the path.* For each task, the agent gets an extrinsic reward of +1 from the environment for reaching the goal, and an extrinsic reward of zero in all other cases.

For each of the tasks, we generate a reference trajectory, and use Amazon Mechanical Turk to obtain 3 descriptions for the trajectory. We use each of these descriptions as language commands in our MDP+L experiments, as described below. Note that we do not use the reference trajectories to aid learning the policy in MDP+L; they are only used to collect language commands to be used in our experiments.

For all experiments, the policy was trained using Proximal Policy Optimization, a popular on-policy RL algorithm [Schulman et al., 2017], for 500,000 timesteps.

3.4.1 How Much Does Language Help?

To evaluate how adding language-based rewards help with policy training, we experiment with 2 different RL setups as follows:

1. **ExtOnly**: In this setup, we use the original environment reward, without using

*Although the tasks (and corresponding descriptions) involve interactions with objects, we observe that just using actions, as we do in our approach, already gives improvements over the baseline, likely because most objects can be interacted with only in one way.

language-based reward. This is the standard MDP setup, and serves as our baseline.

2. **Ext+Lang**: In this setup, in addition to the original environment reward that the agent gets on completing the task successfully, we also provide the agent potential-based language reward R_{lang} at each step, as described in Section 3.2.2.

The following metrics were used for comparing the aforementioned settings:

1. **AUC**: From each policy training run, we plot a graph with the number of timesteps on the x-axis and the number of successful episodes on the y-axis. The area under this curve is a measure of how quickly the agent learns, and is the metric we use to compare two policy training runs.

2. **Final Policy**: To compare the final learned policy with ExtOnly and Ext+Lang, we perform policy evaluation at the end of 500,000 training steps. For each policy training run, we use the learned policy for an additional 10,000 timesteps without updating it, and record the number of successful episodes.

For the Ext+Lang setup, we perform validation over the three types of language encoders described in Section 3.2.2 (InferSent / GloVe+RNN / RNNOnly). Further, we define the joint reward function as $R_{total} = R_{ext} + \lambda R_{lang}$. The type of language encoder and the hyperparameter λ are selected using a validation set.

Results

At test time, we performed 10 policy learning runs with different initializations for each task and each description. The results, averaged across all tasks and descriptions, are

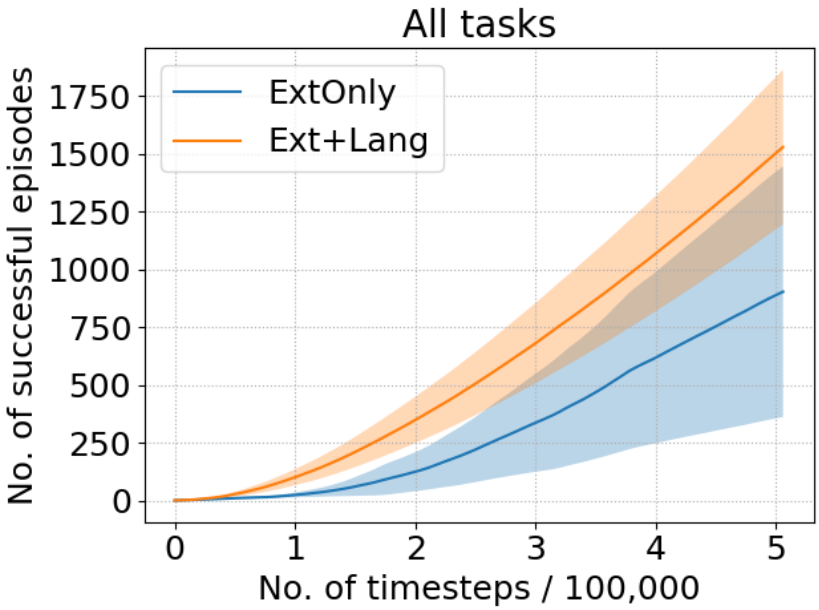


Figure 3.5: Comparison of different reward functions: The solid lines represent the mean successful episodes averaged over all tasks, and the shaded regions represent 95% confidence intervals.

summarized in Figure 3.5, from which we can conclude that Ext+Lang learns much faster than ExtOnly, demonstrating that using natural language instructions for reward shaping is effective. In particular, the average number of successful episodes for ExtOnly after 500,000 timesteps is 903.12, while Ext+Lang achieves that score only after 358,464 timesteps, which amounts to a 30% speed-up. Alternately, after 500,000 timesteps, Ext+Lang completes 1529.43 episodes on average, compared to 903.12 for ExtOnly, thereby giving a 60% relative improvement.

Statistical Significance Tests

For each task, we perform an unpaired t-test between 10 runs of policy training with random initializations using ExtOnly reward function and 30 runs of policy training with random initializations using Ext+Lang reward function (3 descriptions $\times$ 10 runs per description), for both the metrics.

1. **AUC**: Of the total 15 tasks, Ext+Lang gives statistically significant improvement in 11 tasks, leads to statistically significant deterioration in 1 task, and makes no statistical difference in the remaining 3 tasks. This agrees with the conclusions from Figure 3.5, that using language-based reward improves the efficiency of policy training on average.

2. **Final Policy**: We observe that the number of successful episodes for the final policies is statistically significantly greater for Ext+Lang compared to ExtOnly in 8 out of 15 tasks, while the difference is not significant in the remaining 7 tasks. Further, averaged across all tasks, the number of successful episodes with Ext+Lang is more than twice that with ExtOnly. These results suggests that using natural language for reward

Task Id	Description	Correlation coefficients of different actions							
		NO-OP	JUMP	UP	RIGHT	LEFT	DOWN	JUMP-RIGHT	JUMP-LEFT
4	climb down the ladder	-0.60	-0.58	-0.59	-0.61	-0.55	0.07	-0.57	-0.56
	go down the ladder to the bottom	-0.58	-0.58	-0.58	-0.60	-0.53	0.09	-0.59	-0.60
	move on spider and down on the lader	-0.58	-0.54	-0.59	-0.60	-0.49	0.10	-0.58	-0.56
6	go to the left and go under skulls and then down the ladder	-0.37	-0.40	-0.49	-0.43	0.33	0.16	-0.46	-0.01
	go to the left and then go down the ladder	-0.24	-0.26	-0.35	-0.31	0.28	0.36	-0.34	-0.04
	move to the left and go under the skulls	-0.16	-0.25	-0.60	-0.48	0.27	-0.63	-0.52	-0.40
14	Jump once then down	0.00	0.07	-0.15	-0.13	0.51	0.50	0.09	0.52
	go down the rope and to the bottom	-0.03	0.10	-0.16	0.56	0.54	0.33	0.28	0.01
	jump once and climb down the stick	0.11	0.11	0.06	0.04	0.14	0.40	0.25	0.11

Table 3.2: Analysis of language-based rewards

shaping often helps learn a better final policy, and rarely (if ever) results in a worse policy.

Thus, using rewards generated using LEARN often result in both faster policy training, and a better final policy.

3.4.2 Analysis of Language-based Rewards

In order to analyze if the language-based rewards generated from LEARN actually correlate with language descriptions for the task, we compute the Spearman's rank correlation coefficient between each component of the action-frequency vector and corresponding prediction from LEARN over the 500,000 timesteps of policy training. Correlation coefficients averaged across 10 runs of policy training for some selected tasks are reported in Table 3.2.

Figure 3.6 shows the policy training curves for these selected tasks. This analysis supports some interesting observations:

1. For task 4 with simple descriptions, only the DOWN action is positively correlated with

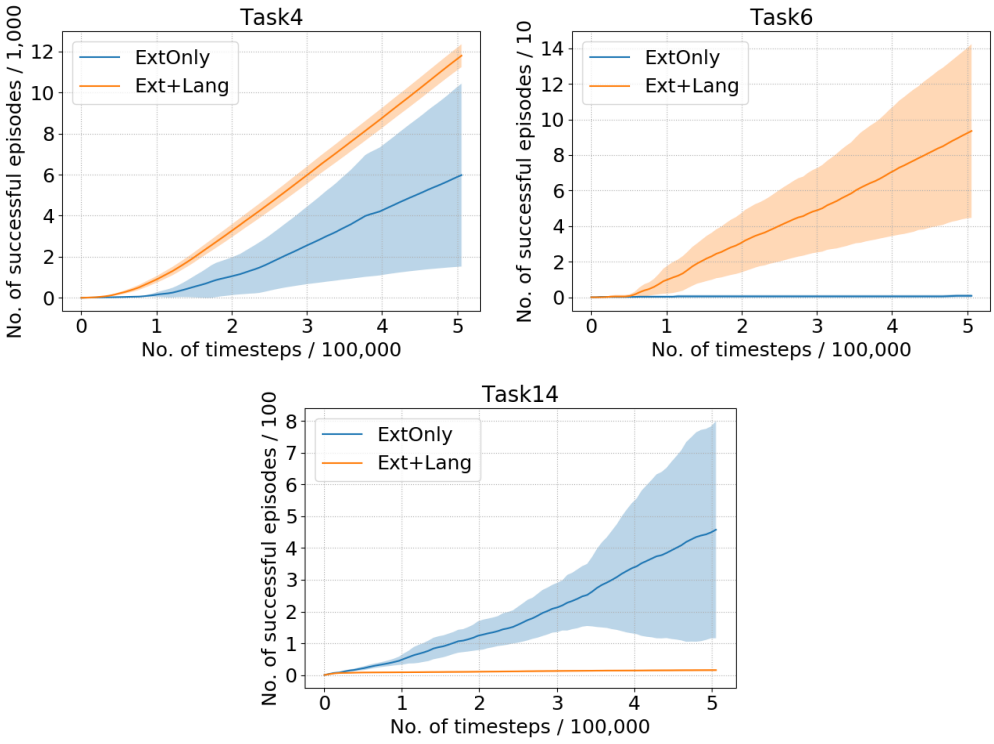


Figure 3.6: Comparisons of different reward functions for selected tasks

language-based reward. All other actions have a strong negative correlation with language-based reward, suggesting that the proposed approach discourages those actions, thereby guiding exploration towards relevant actions.

2. For task 6 with more complex descriptions, LEARN correctly predicts language rewards to be correlated with actions `LEFT` and `DOWN`. For the third description, since the description does not instruct going down, the language reward is negatively correlated with the `DOWN` action. Indeed, we notice in our experiments that we obtain statistically significant improvement in AUC for the first two descriptions, while no statistically significant difference for the third description.

3. Task 14 represents a failure case. Language rewards predicted by LEARN are not well-correlated with the description, and consequently, using language-based rewards results in statistically significant deterioration in AUC. In general, we observe that groundings produced by LEARN for descriptions involving the word "jump" are noisy. We hypothesize that this is because (i) the `JUMP` action typically appears with other actions like `LEFT` and `RIGHT`, and (ii) humans would typically use similar words to refer to `JUMP`, `JUMP-LEFT` and `JUMP-RIGHT` actions. These factors make it harder for the network to learn correct associations.

Note that LEARN does not see action names used in Table 3.2 (`NO-OP`, `JUMP`, etc.); instead, actions are represented as ordinals from 0 through 17. Thus, we see that our approach successfully learns to ground action names to actions in the environment.[†]

[†]While there are a total of 18 actions, we only report the most common 8 actions in Table 3.2 for brevity. The omitted 10 actions jointly constitute less that 1% of the actions in the training data.

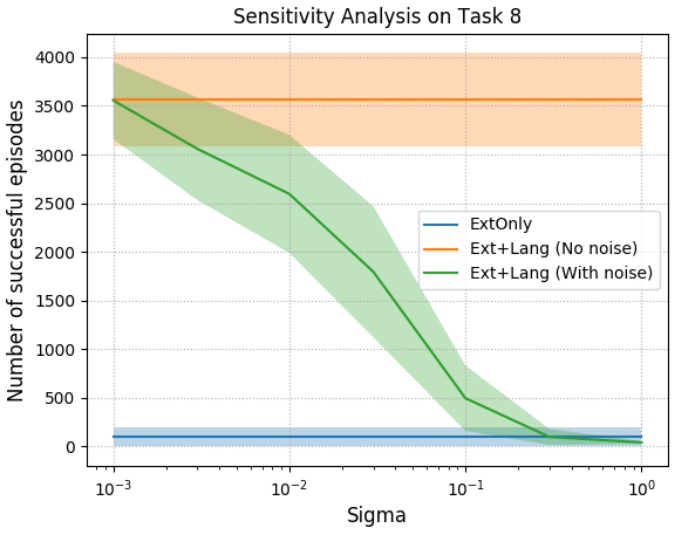


Figure 3.7: Effect of adding noise to the predictions of LEARN.

3.4.3 Sensitivity Analysis

To better understand the relation between the LEARN module and RL, we added varying amounts of noise to the output of LEARN. Specifically, Gaussian noise $\mathcal{N}(0, \sigma)$ was added to the potential function as described in Section 3.2.2, where σ was varied from 0.01 to 1.0. The results for Task 8 are shown in Figure 3.7, from which we can see that the language-based rewards improve over the baseline even with significant amounts of noise. This suggests that the predictions from the LEARN module are fairly robust.

3.5 Conclusions

We proposed a novel framework to address the high sample complexity of policy training in sparse reward settings, by using natural language, which is easy to provide for non-expert users. Our method first learns a neural network to predict the relatedness between

the action frequencies of a trajectory, and a linguistic description of the task, using supervised learning. The outputs of this neural network are then used to define shaping rewards, in addition to sparse rewards from the environment. We prove that these rewards do not change the optimal policy, and our experiments on a diverse set of tasks in the Atari game Montezuma's Revenge show that using these language-based rewards lead to both faster policy training, and a better policy on average. Further, our analyses show that LEARN is able to effectively ground natural language to actions in the environment, and our method is fairly robust to noise in the language-based rewards.

Chapter 4

Guiding Reinforcement Learning Using Natural Language by Mapping Pixels to Rewards

4.1 Introduction

The approach presented in Chapter 3 is a simple and effective way to generate intermediate rewards for sample-efficient learning in sparse reward settings. However, it suffers from several limitations—(1) the action frequency vector can only be defined for discrete action spaces, (2) by only looking at the frequency of actions, the temporal information in the trajectories is discarded, and (3) the model only uses the actions, ignoring the information in the states.

For instance, consider the domain shown in Figure 4.1, which is adapted from the Meta-World benchmark [Yu et al., 2019]. Here, we want the robot to press the green button. Different tasks in this domain require interacting with different objects in the presence of a few other distractor objects.

Since linguistic descriptions of such tasks would typically be in terms of the object to be interacted with, whose positions could change across different scenarios, learning a relatedness model between actions and language without taking into account the state (i.e. the image) will not generalize across scenarios. Thus, we extend prior work to learn a relatedness model between *sequences of states* and linguistic descriptions, and show that

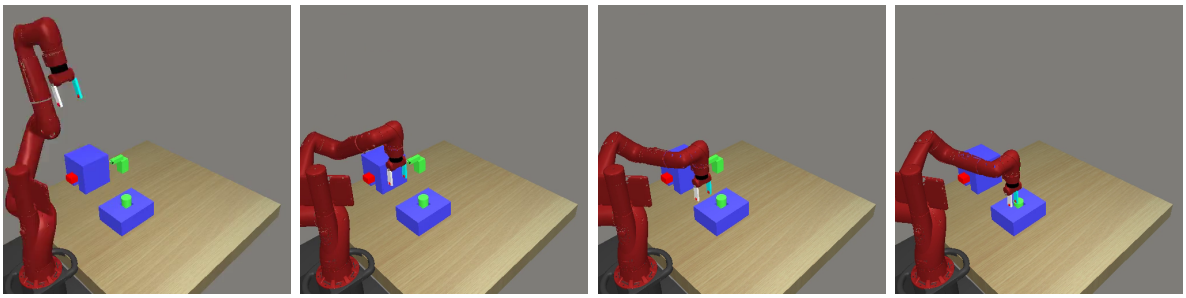


Figure 4.1: A simulated robot completing a task ("push the green button") in the Meta-World domain.

generating language-based scores from the resulting model improves the efficiency of policy training on unseen scenarios.

Using the sequence of states instead of the frequency of actions poses several challenges. First, unlike action frequencies, which are low-dimensional and discard most redundant information in the trajectories, sequences of states are high dimensional, and contain a lot of non-discriminatory information, making them prone to overfitting. Thus, a more careful data preprocessing might be required. Another issue with using states is that a single viewpoint might make learning harder due to perceptual aliasing and occlusions. We address this by feeding multiple viewpoints of the current state to the model. Finally, for the relatedness model to be effective during policy training, it must work with incomplete trajectories. While action frequencies extracted from partial trajectories are similar to those extracted from complete trajectories (particularly when the action space is small, as in prior work), the sequence of states for partial trajectories might be much harder to classify compared to the full sequence of states (for example, because the object being manipulated is only interacted with towards the end of the trajectory). We modify the supervised training approach from prior work to address these challenges.

Our experiments on a diverse set of tasks in the Meta-World domain [Yu et al., 2019] demonstrate that the proposed approach results in improved sample efficiency during policy learning, both in sparse and hand-designed dense reward settings. This motivates a new paradigm where language could be used to improve over hand-designed rewards, which may be suboptimal owing to the difficulty of designing rewards by hand. Our experiments highlight several useful properties of our approach. First, as mentioned above, our approach learns an association between language and trajectories in the environment purely from data, without any assumptions about the structure of the environment or the language. Nor does it require hand-engineering of features, which is difficult to scale as the number of objects and the variation in linguistic descriptions grow. Additionally, the relatedness model is agnostic to the end task on which policy training is performed. As such, the supervised training phase is required only once for a given domain, and the resulting model can then be used on any downstream policy training task. Finally, since we generate rewards from the relatedness model for policy training, our approach is compatible with any choice of RL algorithm.

4.2 Approach

We use the MDP+L framework introduced in Section 3.2, defined as $M' = \langle S, A, T, R, \gamma, l \rangle$, where l is an instruction describing the task using natural language, and the other quantities are as defined as in a standard MDP. Further, we also use the same two-phase framework for learning in an MDP+L described in Chapter 3, which we describe again below:

Phase 1: A neural network (PixL2R) is trained to predict whether a given trajectory and language are related or not. This requires paired ⟨trajectory, language⟩ data in the

environment. We describe this phase in detail in Section 4.2.1.

Phase 2: Next, a policy is trained for a new task – in addition to the extrinsic reward from the environment, the agent additionally gets a language command describing the task. At every step, the agent's trajectory so far is compared against the description of the task using the trained PixL2R model and the relatedness scores predicted by the model are used to generate intermediate rewards for reward shaping [Ng et al., 1999]. Section 4.2.2 describes this phase.

Note that the trained PixL2R model can be used during policy learning for a wide variety of downstream tasks, insofar as the objects and linguistic vocabulary in these tasks closely match the data used to train the PixL2R model. Thus, the cost of training PixL2R is amortized across all the downstream tasks.

4.2.1 PixL2R: Pixels and Language to Reward

First, a relatedness model – PixL2R – between a trajectory and a language is trained given paired data using supervised learning, the details of which are described below.

Network Architecture

The inputs to the network consist of a trajectory and a natural language description. Representing the trajectory using a single sequence of frames may be prone to perceptual aliasing and occlusion. Thus, our network architecture is designed to take multiple views as inputs.

Using multiple viewpoints allows us to study the proposed approach when it has complete information about the task. In some real-world settings, such as office or factory

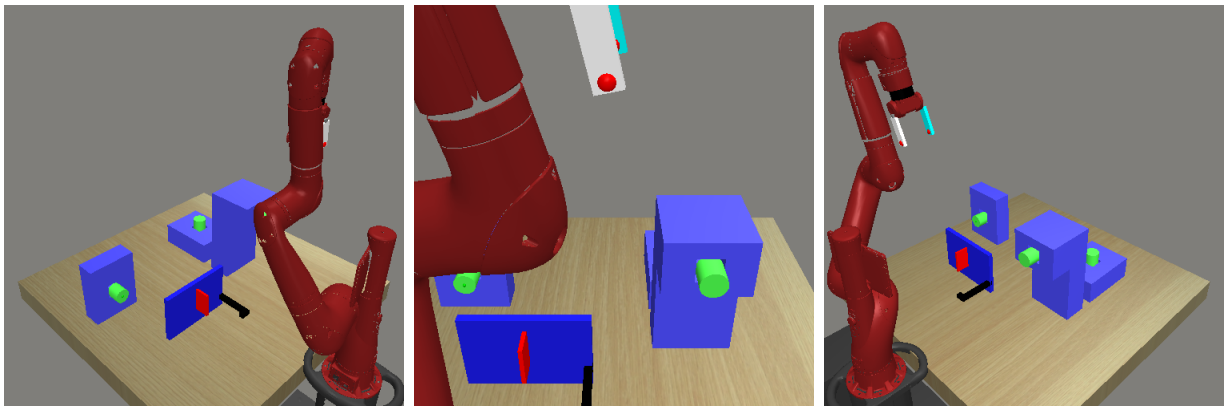


Figure 4.2: Viewpoints used for data collection and experiments.

environments, multiple cameras could be installed to ensure the agents have different viewpoints. Alternately, there are techniques that model perceptual aliasing and occlusion, which could be combined with the approach presented here, to make the system more robust when provided with only a single viewpoint.

We use three different viewpoints in our experiments (see Figure 4.2), but it is straightforward to generalize to more or fewer viewpoints. In our ablation experiments, we compare the model described here with a model that takes a single viewpoint as input.

An independent CNN is used for encoding the sequence of frames from each viewpoint to generate a fixed size representation for each frame. These sequence of vectors are concatenated across the views to generate a single sequence of fixed size vectors, which is then passed through a two-layer LSTM to get an encoding of the entire trajectory.

The language description is converted to a one-hot representation, and passed through an embedding layer, followed by a two-layer LSTM. The outputs of the LSTMs encoding the trajectory and the language are then concatenated, and passed through a sequence of

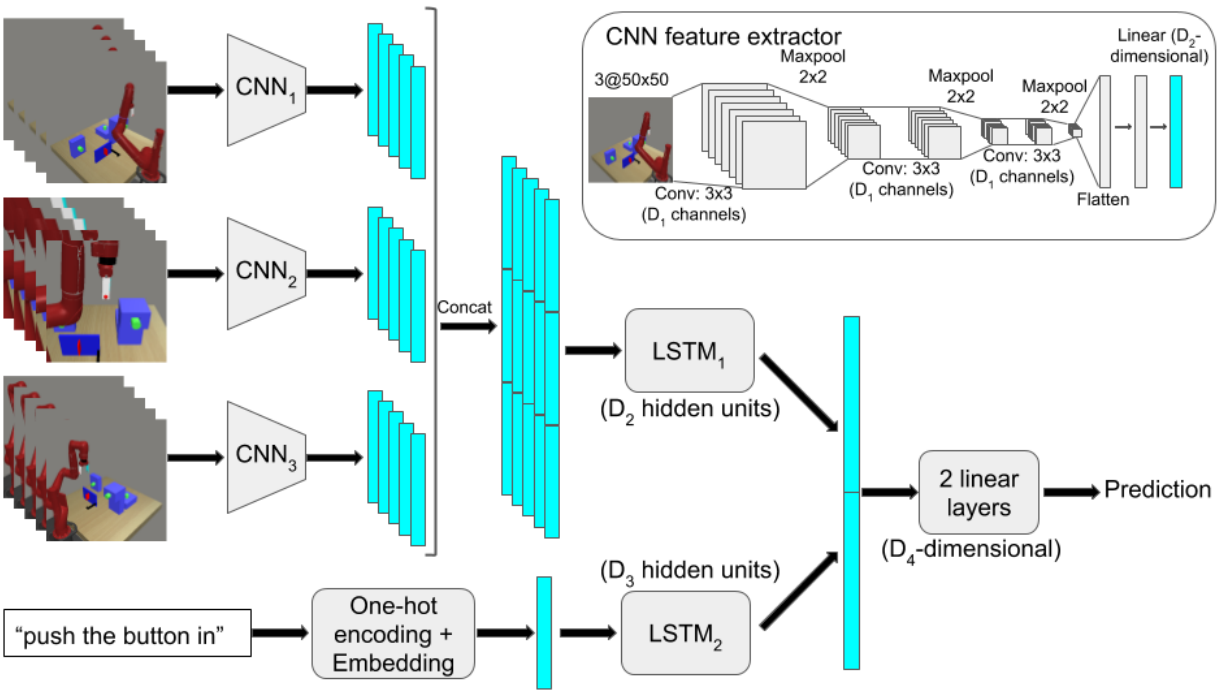


Figure 4.3: Neural network architecture: The sequence of frames from the three viewpoints are passed through three separate CNN feature extractors, and then concatenated across views. The sequence is then passed through an LSTM to obtain an encoding of the trajectory. The given linguistic description is encoded using a randomly initialized embedding layer followed by an LSTM. The outputs of the two LSTMs is concatenated and passed through a sequence of 2 linear layers to generate the final prediction.

fully-connected layers to generate a relatedness score. See Figure 4.3 for a diagram of the neural network.

Data Augmentation

To make the model robust to minor variations in trajectories, as well as incomplete trajectories, we use the following data augmentations.

Frame dropping. For every trajectory in a mini-batch, each frame is independently selected with a probability of 0.1. The resulting sequence of frames is passed through the network. This makes the training faster by reducing the input size, as well as making the network robust to minor variations in trajectories. During policy training, the trajectories are subsampled uniformly to keep 1 frame in every 10.

Partial trajectories. Since during policy training the model will have to make predictions for partial trajectories, we use partial trajectories during supervised learning as well. Given a trajectory of length L, we sample $l \sim Uniform\{1, \ldots, L\}$, and use the first l frames of the trajectory.

Training Objectives

Classification. First, we train the neural network using binary classification, as in Chapter 3. The final output of the network is a two-dimensional vector, corresponding to the logits for the two classes – `RELATED` and `UNRELATED`. The network is trained to minimize the cross-entropy loss.

As mentioned above, we train the model with partial trajectories of different lengths to better match the distribution of trajectories that will be seen during policy learning. However, partial trajectories might sometimes be hard to classify as related or unrelated to the description, since it requires extrapolating the complete path the agent will follow. Our preliminary experiments suggest that these harder to classify examples affect learning—on unseen complete trajectories, a model trained with only complete trajectories has a lower error compared to a model trained on both complete and partial trajectories. This motivated us to experiment with a regression-based objective as an alternative, which we describe next.

Regression. In this setting, the model predicts a scalar relatedness score between the given trajectory and language, which is mapped to $[-1, 1]$ using the tanh() function. The ground truth score is defined as $s \cdot \frac{l}{L}$, where $s = 1$ for `RELATED` and $s = -1$ for `UNRELATED` (trajectory, language) pairs, l is the length of the incomplete trajectory and L is the length of the complete trajectory as described above. Thus, given a description, a complete related trajectory has a ground truth score of 1, while a complete unrelated trajectory has a score of -1. Shorter trajectories smoothly interpolate between these values, with very small trajectories having a score close to 0. The network is trained to minimize the mean squared error. Intuitively, this results in a small loss when the model predicts the incorrect sign on short trajectories. As the trajectories become longer, incorrect sign predictions result in higher losses.

The network is trained end-to-end using an Adam optimizer (Kingma and Ba [2014]). The hyperparameters are tuned on a validation set, using random search with 8 different seeds.

4.2.2 Policy Learning Phase

Having learned a PixL2R model as described above, the relatedness scores from the model can be used to generate language-based intermediate rewards during policy learning on new scenarios. During policy training, the agent receives a natural language description of the goal, in addition to the extrinsic reward from the environment. At every timestep, the PixL2R model is used to score trajectories executed by the agent against the given natural language description, to generate intermediate rewards. As in Chapter 3, we used potential-based shaping rewards [Ng et al., 1999].

For the classification setting, we used the potential function $\phi(s_t) = p_R(s_t) - p_U(s_t)$, where p_R and p_U are the probabilities assigned by the model to the classes `RELATED` and `UNRELATED` respectively. For the regression setting, the relatedness score predicted by the model is directly used as the potential for the state. Note that for both the settings, the potential of any state lies in $[-1, 1]$.

4.3 Domain and Dataset

We use Meta-World [Yu et al., 2019], a recently proposed benchmark for meta-reinforcement learning, which consists of a simulated Sawyer robot and everyday objects such as a faucet, window, coffee machine, etc. Tasks in this domain involve the robot interacting with these objects, such as turning the faucet clockwise, opening the window, pressing the button on the coffee machine, etc. Completing these tasks requires learning a policy for continuous control in a 4-dimensional space (3 dimensions for the end-effector position, and the fourth dimension for the force on the gripper). While the original task suite consists of only one object in every task, we create new environments which contain one or more objects

in the scene, and the robot needs to interact with a pre-selected object amongst those. In a sparse reward setting, the agent is given a non-zero reward only on successfully interacting with the pre-selected object. In the absence of any other learning signal, the agent might have to learn to approach and interact with multiple objects in the scene in order to figure out the correct object. We intend to test whether using natural language to describe the task in addition to the sparse reward helps alleviate this issue.

To create a benchmark for our experiments, 13 tasks were selected from the Meta-World task suite. This gave us a total of 9 objects to interact with (for 4 objects, multiple tasks can be defined, e.g. turning a faucet clockwise or counter-clockwise). We then created 100 scenarios for each task as follows: In each scenario, the task-relevant object is placed at a random location on the table. Then, a newly sampled random location is sampled, and one of the remaining objects is placed at this position. This process is repeated until the new random location is close to an already placed object. This results in 1300 scenarios in total, with a variable number of objects in each scenario.

Next, to obtain descriptions for these tasks, we needed videos for these scenarios. To do this, a policy was trained for each scenario independently using PPO [Schulman et al., 2017], which was then used to generate one video of the robot completing the task in the scenario. For this purpose, we used the dense rewards defined in the original Meta-World benchmark for various tasks. The median length of trajectories across all generated videos is 131 frames. Note that the policies are not used by our algorithm and are only needed to generate the videos for natural language data collection, so the videos could also be generated using human demonstrations.

To collect English descriptions of these tasks, Amazon Mechanical Turk (AMT) was

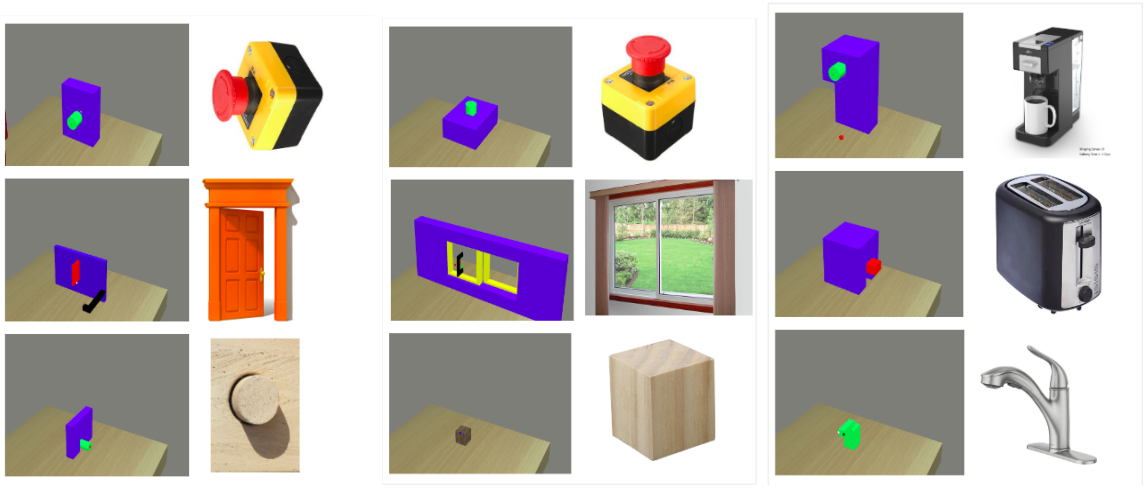


Figure 4.4: List of objects used

used, with a similar setup as described in Section 3.3. Workers were first provided with the instructions and an example trajectory with a description. They were then shown a video and were given 4 possible descriptions to choose from. Only workers that passed this basic test were allowed to provide descriptions for the main tasks.*

Since the models of the objects in the environment are coarse, it is usually non-trivial to recognize the real-world objects they represent from the models alone. To guide the AMT workers to use the names of real-world objects the models represent, we showed a table of the models with prototypical images of real-world objects that closely match the models (shown in Figure 4.4). This enabled us to get descriptions that use the real-world object names, without priming the workers with specific words.†

*The objects used for the example and the test are different from those used in the main tasks.

†Despite using this technique, we still got some responses where people described the models directly instead of using the object names, e.g. "Pull the red box out slightly in blue square." instead of using the word *toaster*.

Each worker was asked to provide descriptions for 5 videos, which were sampled from the 1300 scenarios with the constraint that no two videos involved interacting with the same object. Since using a single viewpoint is susceptible to perceptual aliasing and occlusion as mentioned earlier, we used 3 viewpoints for data collection on AMT, as in our model (Figure 4.2). We used simple heuristics (such as number of words and characters in the descriptions) to automatically filter out clearly bad descriptions. Some examples of descriptions (after filtering) are shown in Table 4.1.

A total of 520 descriptions were collected, which gives us 40 descriptions per task on average. Interestingly, most of the descriptions involve only the object being manipulated, with no reference to other objects in the scene. As such, a description collected for one scenario for a task can be paired with any of the 100 scenarios for the corresponding task.

The distribution of number of words per description is shown in Figure 4.5. The full dataset contains 264 unique words, while the training set contains 248 unique words.

4.4 Experiments

4.4.1 Training the PixL2R model

Given the data collected using the process described in Section 4.3, for each task, 79 scenarios were used for training, 18 for validation, and 3 for testing. Similarly, the descriptions for each task were split as follows—5 for validation, 3 for testing, and the remaining for training (there could be variable number of descriptions per task).

From the ⟨trajectory, language⟩ pairs obtained after data collection, positive examples were generated by pairing a scenario for one of the 13 tasks with a randomly sampled description of the corresponding task. Negative examples were generated by pairing

Task Id	Description
0	Press the button.
0	Pressing the button
1	Push peg in to hole.
1	Push the green button.
2	Turn on the coffee maker
2	push in the green button
3	Push toaster handle down
3	Push down the red block.
4	pressing down the object
4	pull down the red switch
5	move the plate down
5	push down the slider
6	Close the door
6	Open the door.
7	twisting the cube
7	rotate the object
8	Rotate the lever anticlockwise
8	Turn the faucet to the right.
9	rotating the object
9	turn on the faucet
10	Open the window.
10	Open the yellow window.
11	Slide the window to the left.
11	Close the Window.
12	pull out the green block
12	Pull out the green piece

Table 4.1: Examples of descriptions collected using AMT.

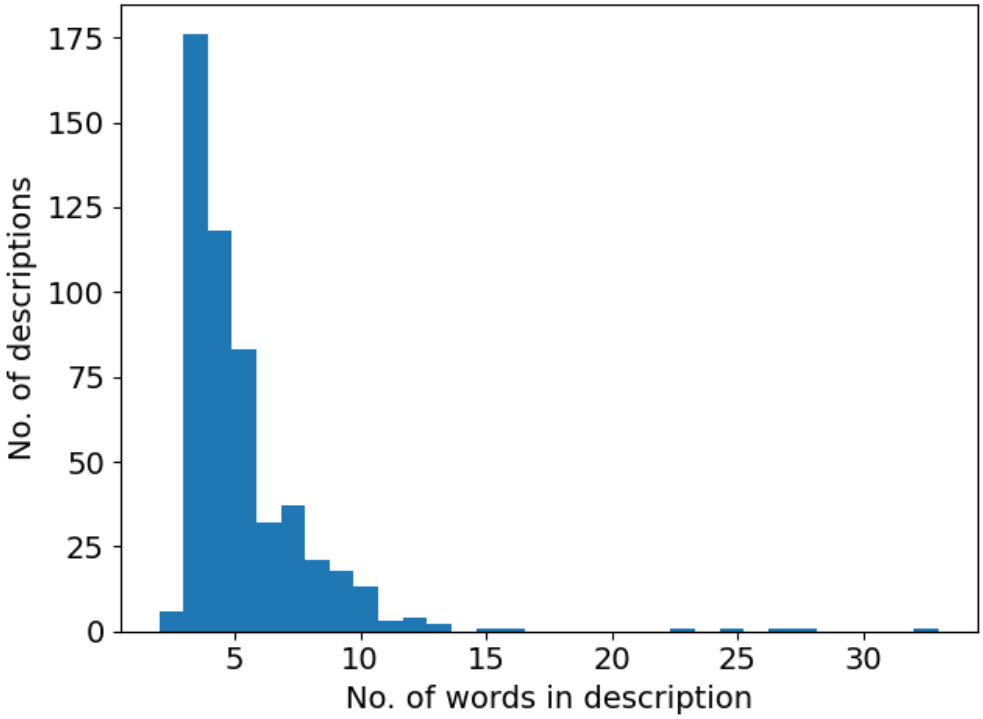


Figure 4.5: Distribution of number of words per description in the collected dataset.

trajectories from a task with descriptions from unrelated tasks as follows. If there was only one object in the scene, then it was paired with the description of any of the remaining 12 tasks. If the scene contained more than one object, then it was paired with the description of the task corresponding to one of the alternate objects in the scene. For instance, if a scenario contains a faucet, a toaster, and a coffee machine, with the toaster being the object to be interacted with, then a negative description for this scenario was chosen from the tasks involving interaction with the faucet or the coffee machine. Using such a scheme for generating negative examples is important because naively creating pairs of trajectories with descriptions of any other task randomly might result in most negative examples lacking the task-relevant object mentioned in the description. As such, the network might learn to use the *presence* of the mentioned object to compute relatedness, instead of whether the mentioned object is being *interacted with*.

4.4.2 Policy Training with Language-based Rewards

To empirically evaluate the effectiveness of PixL2R, the following setup was used. For each of the 13 tasks, a policy was trained for the 3 test scenarios using the PPO algorithm. Each policy training was run for 500,000 timesteps, and the number of successful completions of the task were recorded. The robot's end-effector was set to a random position within a predefined region at the beginning of each episode, and the maximum episode length was restricted to 500 timesteps.

First, policy training was run with 15 random seeds, both in the sparse reward setting (1 if the agent reaches the goal, and 0 otherwise), denoted as `Sparse`, and the hand-designed dense reward setting (defined in the original Meta-World benchmark), denoted as `Dense`.

Then, a Kruskal-Wallis test was used for each scenario to identify scenarios where the number of successful episodes using dense rewards was statistically significantly more than the number of successful episodes using sparse rewards. All subsequent comparisons were done on the 16 (out of 39) scenarios for which this was true. Intuitively, these 16 tasks are too difficult to learn from sparse rewards, while they can be learned using dense rewards. Therefore, language-based dense rewards should be useful on these tasks. The remaining tasks are presumably either too simple that they can be learned with sparse rewards alone, or are too difficult to learn within 500,000 timesteps even with hand-designed dense rewards.

For each of the scenarios, a policy was trained using 4 different reward settings:

1. `Sparse`: The agent gets a reward of 1 for reaching the goal, and 0 in all other cases.
2. `Sparse+RGR`: The agent gets language-based rewards generated by PixL2R trained using the regression objective, in addition to the sparse rewards.
3. `Dense`: The agent gets hand-designed dense rewards defined in the original Meta-World benchmark.
4. `Dense+RGR`: The agent gets language-based rewards generated by PixL2R trained using the regression objective, in addition to the hand-designed dense rewards.

The resulting policy training curves are shown in Figure 4.6. Each curve is obtained by averaging over all runs (16 scenarios $\times$ 15 runs per scenario with different random seeds) for that reward type. The results verify that using language-based rewards in addition to sparse rewards result in higher performance on average than using only sparse ones. More interestingly, we find that using language-based rewards in conjunction with hand-designed

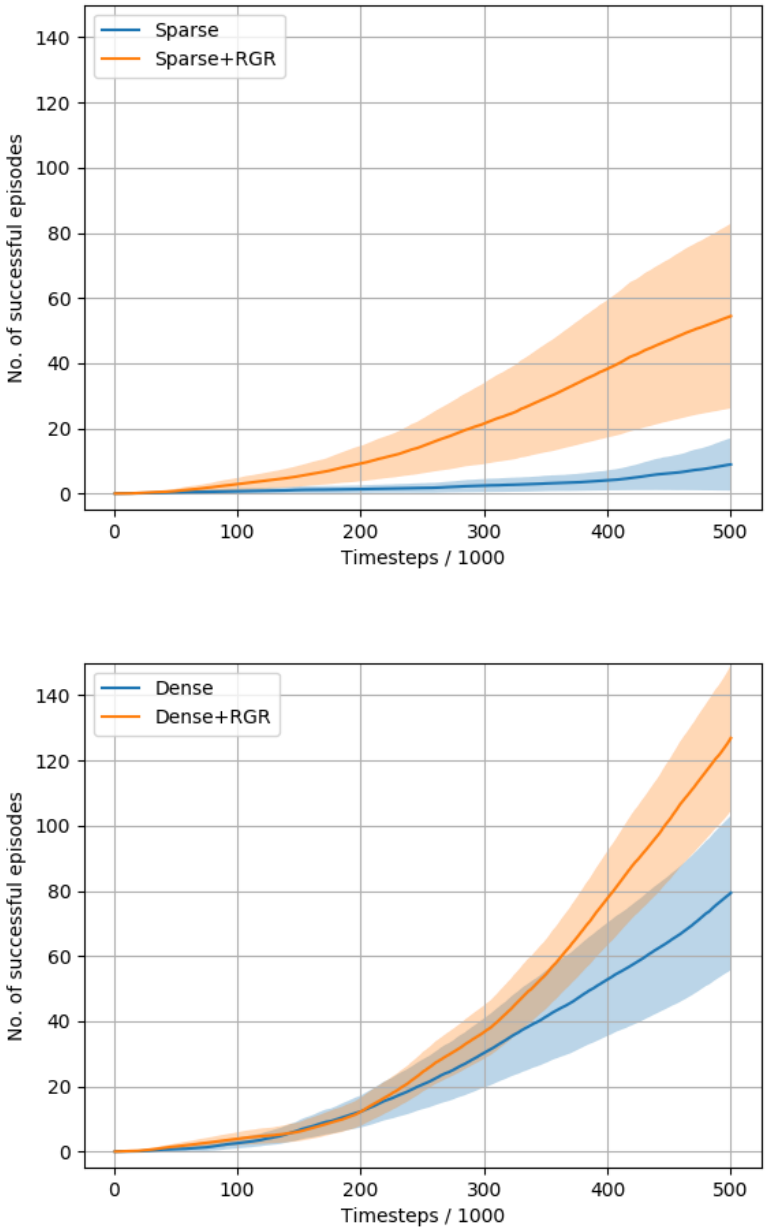


Figure 4.6: A comparison of policy training curves for different reward models. The shaded regions denote 95% confidence intervals.

rewards also results in an improvement. A plausible explanation is that the hand-designed dense rewards in Meta-World are suboptimal, since the reward function for each task consists of several tunable parameters, highlighting the complexity of reward design mentioned earlier. This result motivates a novel paradigm wherein coarse dense rewards could be designed by hand, and the proposed framework can be used to get a further improvement in policy training efficiency by using natural language.

Next, statistical significance was computed to compare the reward functions. For each type of reward, first the average number of successful episodes was computed across all the 15 runs for each scenario, giving 16 mean successful episode scores per reward type. Since the number of successful episodes across different scenarios vary quite a bit, the mean scores for each scenario were scaled to be at most 1, by dividing by the maximum value of the mean score across all reward types for that scenario (including the reward types used in ablation experiments described in Section 4.4.3).

A Wilcoxon signed-rank test was then performed between the sets of normalized scores across reward types. `Sparse+RGR` was found to be statistically significantly better than `Sparse` (p-value=0.007) and `Dense+RGR` was found to be statistically significantly better than `Dense` (p-value=0.034) rewards, at a 5% significance level. Thus, the proposed approach can be used to make policy learning more sample efficient in both sparse and dense reward settings.

4.4.3 Ablations

Having established that policy learning works better with language-based rewards, we ran ablation experiments to better understand our design choices and to inspect what

factors most affect the efficiency of policy learning.

All the ablation experiments were performed with language-based rewards added to dense rewards, since most applications of RL in robotics currently use dense hand-designed rewards (which could be suboptimal for complex tasks).

1. **LastFrame**: To analyze whether using the full sequence of frames contains more information than the last frame, instead of using the sequence of frames in the trajectory, only the last frame of the trajectory was used, both for training the PixL2R model, as well as for policy training.

2. **MeanpoolLang**: To study if the temporal ordering of the words in the description is useful, the LSTM used to encode the language was replaced with the mean-pooling operation.

3. **MeanpoolTraj**: To study if the temporal ordering of the frames in the trajectory was useful, the LSTM used to encode the sequence of frames was replaced with the mean-pooling operation.

4. **SingleView**: To study the impact of perceptual aliasing and/or occlusion when using a single viewpoint, instead of using 3 viewpoints for the trajectory, only 1 viewpoint was used. A model was trained with *each* of the three viewpoints in the supervised

Setting	Mean Successful Episodes	p-value w.r.t. Dense
Dense	79.4	-
Dense+RGR	126.9	0.0340
LastFrame	133.5	0.0114
MeanpoolLang	138.3	0.0004
MeanpoolTraj	78.4	0.9601
SingleView	100.4	0.3789
Dense+CLS	102.0	0.6384

Table 4.2: Comparison of various ablations to the Dense+RGR model.

learning phase, and the model with the best validation score was used for policy learning.

5. **Dense+CLS**: Instead of the regression loss, classification loss was used, to understand the benefit of using regression loss when working with partial trajectories.

For each ablation, the same setup was used as for `Dense+RGR`. – training the PixL2R model with 8 random sets of values of hyperparameters, and choosing the model with the best validation accuracy. This model is used to generate rewards for policy training, for each of the 16 scenarios with 5 random seeds for all the 3 descriptions as before.

The mean successful episodes across all runs are reported in Table 4.2 for each setting. Further, the p-values for Wilcoxon tests between each ablation and the `Dense` rewards is reported, from which we can make the following observations:

- Using only the last frame (`LastFrame`), or using mean-pooling instead of an LSTM to encode the language (`MeanpoolLang`) does not substantially affect policy learning effi-

ciency. In both these cases, the resulting model is still statistically significantly better than `Dense` rewards. Both of these results agree with intuition, since the progress in the task can be predicted using the last frame alone, and since the linguistic descriptions are not particularly complex in the given domain, simply looking at which words are present or absent is often sufficient to identify the task without using the ordering information between the words.

- Using mean-pooling instead of an LSTM to encode the sequence of frames (`Meanpool-Traj`) drastically reduces the number of successful episodes, and results in no statistically significant improvement over `Dense`. Again, this agrees with intuition, since it is not possible to infer the direction of movement of the robot from an unordered set of frames.

- Using a single view instead of multiple views (`SingleView`) results in a smaller increase in the number of successful episodes, which is no longer statistically significant over `Dense`. As mentioned earlier, using frames to represent trajectories requires addressing challenges such as perceptual aliasing and occlusion, and these ablation results suggest that using multiple viewpoints alleviates these issues.

- Using classification loss instead of regression (`Dense+CLS`) also leads to a drop in performance, again making the resulting improvements no longer statistically significant. This is consistent with our initial observation (Section 4.2.1), wherein, the learning problem becomes more difficult due to partial trajectories when the classification loss is used.

It is worth noting that while these ablations agree with intuition, and therefore suggest that the model is extracting meaningful information from trajectories and language descriptions, the performance of these variants depends crucially on the domain. For instance, an environment that is not fully observable in the last frame might show a significant drop in performance when using only the last frame instead of the full trajectory.

4.4.4 Word-level Analysis

In order to understand how the supervised learning phase is using different words in the description, the supervised model was used to make predictions on the test set, and the gradient of the loss was computed with respect to the continuous representation of the words in the descriptions (i.e. after the embedding layer). The mean of the absolute values of these gradients is then a measure of how much the prediction is affected by the corresponding word. The values are reported in Table 4.3, which were scaled so that the maximum value for any description is 1.

First, we observe that for all the descriptions, the words describing the main object have a very high average gradient magnitude – *green* and *button* in description 1, *red* and *block* in description 2, *lever* and *toaster* in description 3, *faucet* in description 4, *green* and *lever* in description 5, and *window* in description 6. Several verbs also have a high average gradient magnitude – *turn on* in description 4 and *open* in window. Verbs in other descriptions do not have a high gradient magnitude because for those descriptions, the object affords only one possible interaction, thus making the verb less discriminatory. For the objects *faucet* and *window*, there are two possible actions each (*turning the faucet on or off* and *opening or closing the window*); thus the verb also carries useful information for these

	Descriptions Average magnitude of gradient for each word						
1.	push	the	green	button			
	0.53	0.30	1.00	0.94			
2.	push	down	the	red	block		
	0.42	0.57	0.34	1.00	0.91		
3.	pull	down	the	lever	on	the	toaster
	0.16	0.31	0.15	0.75	0.58	0.36	1.00
4.	turn	on	the	faucet			
	0.94	1.00	0.44	0.87			
5.	slide	the	green	lever	to	the	left
	0.52	0.23	0.94	1.00	0.77	0.30	0.78
6.	open	the	window				
	0.83	0.32	1.00				

Table 4.3: Average magnitude of gradients for different words in a description for the relatedness score prediction.

objects.

This analysis suggests that the model learns to identify the most salient words in the description that are useful to predict the relatedness between a trajectory and language.

4.5 Conclusions

We proposed an extension of the LEARN model, which addresses its key limitations, by learning a relatedness model—PixL2R—between sequence of states and linguistic descriptions. Our experiments on a diverse set of robot manipulation tasks on the Meta-World domain show that the learned model results in statistically significant improvement over the no-language baseline, both in sparse and dense reward settings, motivating a new paradigm for RL, wherein coarse rewards could be designed by hand, and additional sample efficiency

could be obtained during policy training by using language-based rewards. Further, word-level analysis of our approach reveals that the model is able to learn the most salient words in task descriptions.

Chapter 5

Zero-shot Task Adaptation Using Natural Language

5.1 Introduction

In Chapters 3 and 4, we proposed approaches that use natural language to generate intermediate rewards for reinforcement learning, resulting in more efficient policy learning. As described in Chapter 2, another paradigm to teach new tasks to learning agent in sequential decision making problems is using imitation learning [Argall et al., 2009]. This involves showing demonstration(s) of the desired task to the agent, which can then used by the agent to infer the demonstrator's intent, and hence, learn a policy for the task. However, for each new task, the agent must be given a new set of demonstrations, which is not scalable as the number of tasks grow, particularly because providing demonstrations is often a cumbersome process. We propose using language to reduce the burden of providing demonstrations.

To this end, we propose a novel setting—given a demonstration of a task (the *source task*), we want an agent to complete a somewhat different task (the *target task*) in a **zero-shot** setting, that is, without access to *any* demonstrations for the target task. The difference between the source task and the target task is communicated using language.

For example, consider an environment consisting of objects on different shelves of an organizer, as shown in Figure 5.1. Suppose the **source task** (top row) requires moving the green flat block from bottom-right to bottom-left, the blue flat block from middle-left

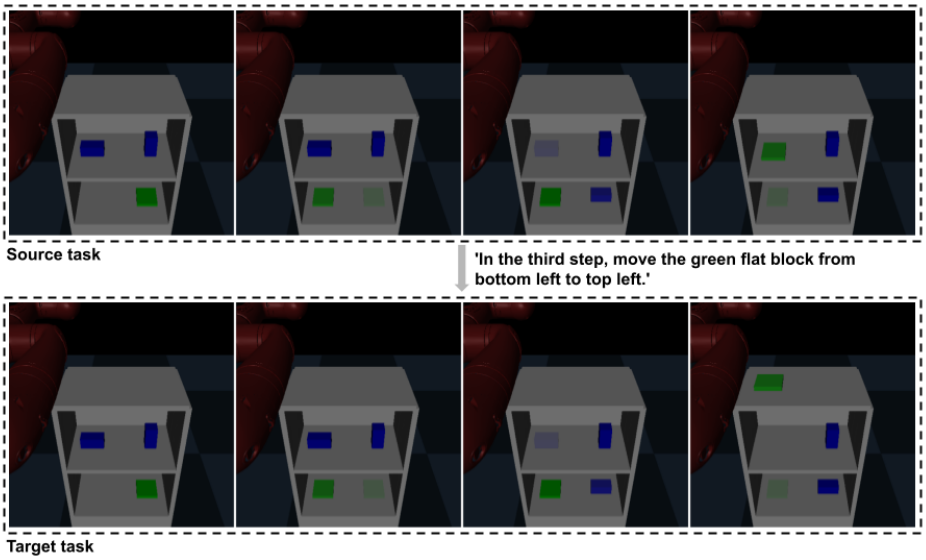


Figure 5.1: Example of the setting: The top row shows the *source task*, while the bottom shows the *target task*. Given a demonstration of the source task, and a natural language description of the difference between the two tasks such as "In the third step, move the green flat block from bottom left to top left.", our goal is to train an agent to perform the target task *without* any demonstrations of the target task.

to bottom-right, and then the green flat block from bottom-left to middle-left. The **target task** (bottom row) is similar, but in the final step, the green flat block should be moved to top-left instead. We posit that given a demonstration for the source task, and a free-form natural language description of the difference between the source and the target tasks, such as "In the third step, move the green flat block from bottom left to top left", an agent should be able to infer the goal for the target task. We propose a framework that can handle a diverse set of adaptations between the source and the target tasks, such as a missing step, an extra step, and swapping the final positions of two objects.

The environment has a similar structure to several real-world applications, where task adaptation using language could be particularly beneficial. For instance, consider the goal of building service robots for home environments. These robots must be able to learn a wide variety of tasks from non-expert users. Many tasks, such as cooking or assembly, involve a sequence of discrete steps, and such tasks could have several variations, like different cooking recipes or assembling different kinds of furniture. Being able to demonstrate one (or a few) of these tasks, and then communicating the difference between the demonstrated task(s) and other similar tasks could significantly reduce the burden of teaching new skills for the users.

These problems involve planning/control at 2 levels—high-level planning over the steps, and low-level control for executing each step. Since our proposed algorithm focuses on the high-level planning, we illustrate our approach on the simple environment shown in Figure 5.1, where the low-level control is abstracted away. However, our framework is general, and can be combined with approaches that perform low-level control.

The proposed setting is challenging for several reasons. First, most existing approaches in imitation learning and instruction-following infer the goal for a target task from

a demonstration or an instruction, respectively. However, in our setting, neither of these modalities is sufficient by itself, and the agent must be able to combine complementary information from the source demonstration(s) and the natural language descriptions, in order to infer the goal for the target task. Second, as in Chapters 3 and 4, the agent must be able to perform symbol grounding [Harnad, 1990], that is map concepts in the description to objects and actions. Finally, in order to be scalable, we intend to learn a purely data-driven model that can does not require engineering features for the language or the environment, and can learn to infer the goal for the target task directly from data.

It is worth noting that imitation learning is often used in conjunction with reinforcement learning, for instance to initialize the policy using a set of demonstrations [Hester et al., 2018]. Thus, the approach we present here for using natural language to aid imitation learning can be combined with the approaches presented in Chapters 3 and 4, to develop systems that use both RL and IL aided by language.

Further, since the proposed setting requires combining information from both the demonstration and the language, it can therefore serve as an important step towards building systems for more complex tasks which are difficult to communicate using demonstrations or language alone.

We introduce the Language-Aided Reward and Value Adaptation (LARVA) model that takes in a dataset of source demonstrations, linguistic descriptions, and either the reward or optimal value function for the target task, and is trained to predict the reward or optimal value function of the target task given a source demonstration and a linguistic description. The choice between reward and value prediction could be problem-dependent—for domains with complex transition dynamics, learning a value function requires reasoning

about these dynamics, and therefore, it might be better to use LARVA for reward prediction, with a separate policy learning phase using the predicted rewards; for domains with simpler dynamics, a value function could be directly predicted using LARVA, thereby removing the need for a separate policy learning phase.

5.2 Problem Definition

Consider a **goal-based task**, which can be defined as a task where the objective is to reach a designated goal state in as few steps as possible. It can be expressed using the standard Markov Decision Process (MDP) formalism, as $M = \langle S, A, P, g \rangle$, where S is the set of all states, A is the set of all actions, $P : S \times A \times S \to [0, 1]$ is the transition function, and $g \in S$ is the unique goal state.

The reward function for a goal-based task can be defined as $R(s) = \mathbb{1}[s = g]$, where $\mathbb{1}[\cdot]$ is the indicator function. Thus, for $\gamma < 1$, an optimal policy for a goal-based task must reach the goal state g from any state $s \in S$ in as few steps as possible.

Further, $V^*_R : S \to \mathbb{R}$ denotes the optimal value function under the reward function R, and can be used to act optimally.

Let $\mathcal{T} = \{M_i\}_{i=1}^{N}$ be a **family** of goal-based tasks M_i, each with a distinct goal g_i, and the reward function R_i defined as above. The set of states S_i and actions A_i, the transition functions P_i, and the discount factors γ_i across different tasks may be related or unrelated [Kroemer et al., 2019].

For instance, in the environment shown in Figure 5.1, a goal-based task consists of arranging the objects in a specific configuration defined by a goal state g, while $\mathcal{T}$ is the set

of all multi-step rearrangement tasks in the environment.

Let T_{src}, $T_{tgt} \in \mathcal{T}$ be two tasks, and L be a natural language description of the difference between the tasks. Given a demonstration for the source task T_{src}, and the natural language description L, our objective is to train an agent to complete the target task T_{tgt} in a **zero-shot setting**, i.e., without access to the reward function or demonstrations for the target task.

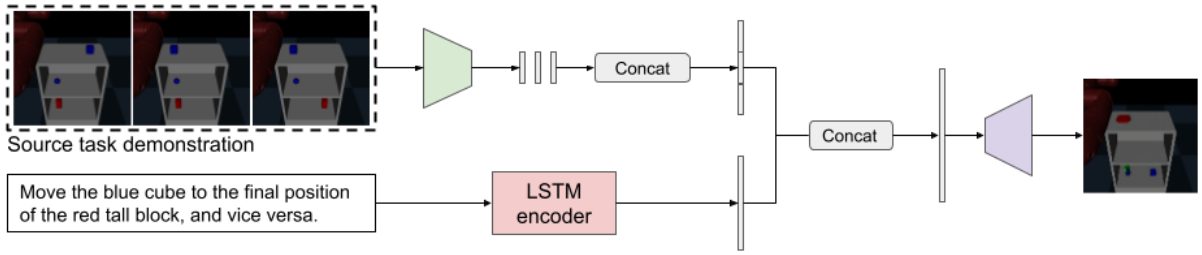
5.3 LARVA: Language-Aided Reward and Value Adaptation

We propose Language-Aided Reward and Value Adaptation (LARVA), a neural network that takes in a source demonstration, τ_{src}, the difference between the source and target tasks described using natural language, l, and a state from the target task, $s \in S_{tgt}$, and is trained to predict either $R_{tgt}(s)$, the reward for the state s in the target task, or $V^*_{R_{tgt}}(s)$, the optimal value of the state s under the target reward function R_{tgt}.

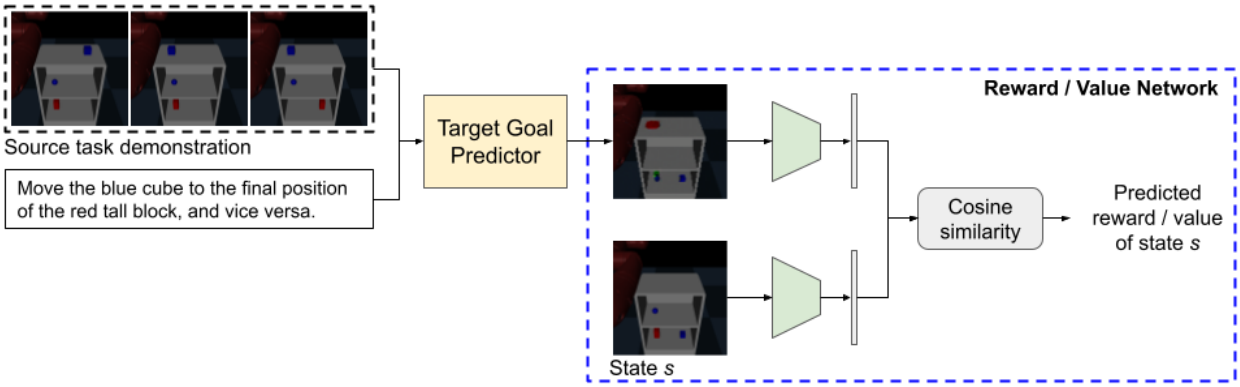
We assume access to a training set $\mathcal{D} = \{(\tau^i_{src}, l^i, g^i_{tgt})\}_{i=1}^N$, where τ^i_{src} is a demonstration for the source task of the i^{th} datapoint, l^i is the natural language description of the difference between the source task and the target task for the i^{th} datapoint, and g^i_{tgt} is the goal state for the target task. The details of the dataset and the data collection process are described in Section 5.4.

5.3.1 Network Architecture

We decompose the learning problem into two subproblems: (1) predicting the goal state for the target task given the source demonstration and the language, and (2) predicting the reward / value of state s given the goal state of the target task. As such, we propose



(a) The target goal predictor takes in a source demonstration and a description to predict the goal state for the target task.



(b) The reward / value network takes the predicted goal state from the target goal predictor, and another state s from the target task to predict the reward or value of the state s under the target reward function.

Figure 5.2: Neural network architecture for LARVA

a neural network architecture that consists of two modules: (1) Target Goal Predictor, and (2) Reward / Value Network (see Figure 5.2). This decomposition allows for additional supervision of the target goal predictor, using the ground-truth goal state for the target task.

5.3.1.1 Target Goal Predictor

Given a sequence of images representing a source demonstration (τ_{src}), and a natural language description of the difference between the source and the target task (L), the target goal predictor module is trained to predict the goal state of the target task (g_{tgt}).

Demonstration Encoder. Each image in the source demonstration is first passed through a convolutional neural network to obtain a D_1-dimensional vector representation, where D_1 is a hyperparameter tuned using the validation set. The resulting sequence of vectors is padded to the maximum demonstration length (T_{max}) in the training data, and the vectors are then concatenated to obtain a single $T_{max} \cdot D_1$-dimensional vector.

Language Encoder. The natural language description is first converted into a one-hot representation using a vocabulary (see Sections 5.4.2 for details about the vocabulary), which is then passed through a two-layer LSTM module to obtain a vector representation of the description. The hidden size of the LSTM is set to D_2, which is tuned using the validation set.

Target Goal Decoder. The vector representations of the source demonstration and the natural language description are concatenated, and the resulting vector is passed through a

deconvolution network to obtain an image representation $\hat{g}$ of the target goal state.

5.3.1.2 Reward / Value Network

The reward or value network takes the predicted target goal state $\hat{g}$ and another state s from the target task as inputs, and is trained to predict the reward or the value respectively of state s under the target reward function. The predicted goal state $\hat{g}$ and the state s are encoded using the same CNN encoder (i.e. shared weights) used for encoding the demonstration frames in the target goal predictor, to obtain D_1-dimensional vector representations. The reward or value of state s is computed as the cosine similarity between the vector representations of $\hat{g}$ and the state s. We represent the ground-truth reward or value as $f(s)$, while the network prediction as $\hat{f}(s)$.

5.3.2 Training

To train the model, we assume access to a dataset $\mathcal{D} = \{(\tau^i_{src}, l^i, g^i_{tgt})\}_{i=1}^N$. Using the goal state for the i^{th} target task, the reward function $R^i_{tgt}(s) = \mathbb{1}[s = g^i_{tgt}]$, and the corresponding optimal value function for the target task can be computed, which is used to supervise the model training as described below.

Training Objectives

Mean Squared Error. Since we want the model to regress to the true reward or value $f(s)$ for states $s \in S^i_{tgt}$, a natural choice for the loss function is the mean squared error (MSE),

$$\mathcal{L}_{\text{f}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{s \in S^i_{tgt}} (f(s) - \hat{f}(s))^2 \tag{5.1}$$

Target Goal Supervision. Further, we additionally supervise the Target Goal Predictor using the true goal state g^i_{tgt} for the i^{th} target task, using an MSE loss,

$$\mathcal{L}_{\text{goal}} = \frac{1}{N}\sum_{i=1}^{N}(g^i_{tgt} - \hat{g}^i_{tgt})^2 \tag{5.2}$$

Thus, the overall training loss is given by $\mathcal{L} = \mathcal{L}_{\text{f}} + \lambda\mathcal{L}_{\text{goal}}$, where λ is a hyperparameter tuned using a validation set.

Optimization

For training the model, a datapoint $(\tau^i_{src}, l^i, g^i_{tgt})$ is sampled uniformly at random from $\mathcal{D}$. When predicting the value function, a target state s is sampled uniformly at random from S^i_{tgt} at each step of the optimization process. When predicting the reward function, the goal state g^i_{tgt} is sampled with 50% probability, while a non-goal state is sampled uniformly at random otherwise. This is required because the reward function is sparse. We use an Adam optimizer Kingma and Ba [2014] to train the network end-to-end for 50 epochs, with weights initialized randomly according to Glorot and Bengio [2010]. A validation set is used to tune hyperparameters via random search.

5.4 Environment and Dataset

In this section, we describe the environment we use in our experiments. While the framework described above is fairly general, in this work, we focus on a simpler setting that is more amenable to analysis. Specifically, we assume discrete state and action spaces S and A, and deterministic transitions, i.e., $P(s, a, s') \in \{0, 1\}, \forall (s, a, s') \in S \times A \times S$.

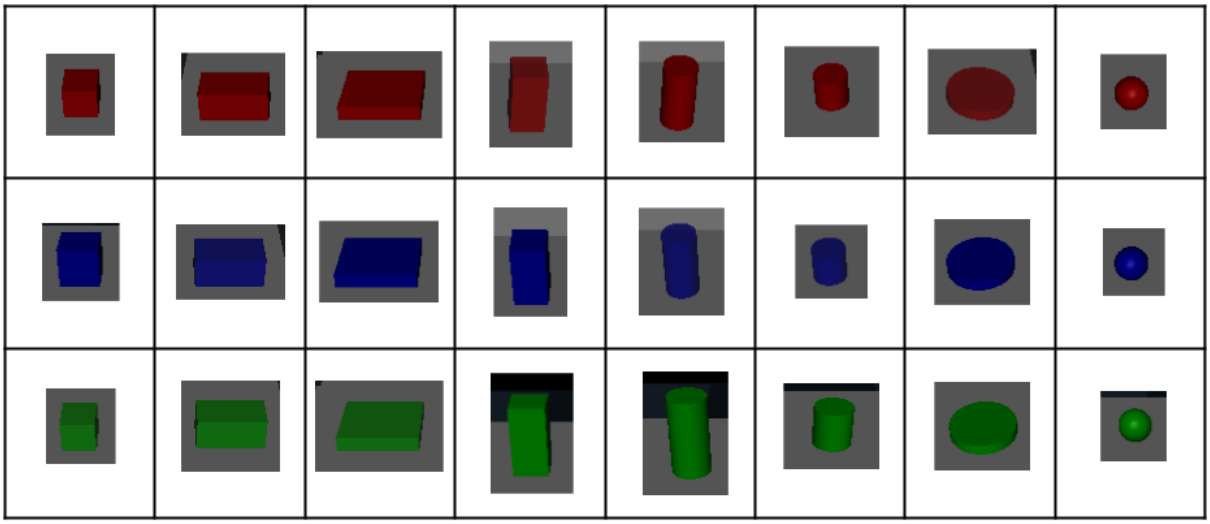


Figure 5.3: Objects in the Organizer Environment

5.4.1 The Organizer Environment

We propose the Organizer Environment, which consists of an organizer with 3 shelves. There are 8 distinct objects, and each object can take one of 3 colors (red, blue, or green), giving a total of 24 distinct colored objects. Figure 5.3 shows all the objects in the Organizer environment.

Objects can be placed in each shelf, either to the left or the right, resulting in a total of 6 distinct positions, `POSITIONS` = {`Top-Left`, `Top-Right`, `Middle-Left`, `Middle-Right`, `Bottom-Left`, `Bottom-Right`}.

Objects can be placed in different configurations to create different states. In our experiments, we use tasks with 2 or 3 objects. The total number of states with 2 or 3 objects (i.e. $|\bigcup_{T\in\mathcal{T}} S_T|$) is 285,120. The action space A is common across all tasks, and consists of 30 move actions, $\texttt{MOVE}(p_i, p_j), p_i, p_j \in \texttt{POSITIONS}, p_i \neq p_j$. Finally, there is a `STOP` action

that indicates the termination of an episode.

5.4.2 Language Data

In this work, we experiment with 6 types of adaptations: (1) moving the same object in the source and target tasks, but to different final positions; (2) moving a different object in the source and target tasks, but to the same final position; (3) moving two objects in the source and target tasks, with their final positions swapped in the target task; (4) deleting a step from the source task; (5) inserting a step to the source task; and (6) modifying a step in the source task. Table 5.1 shows an example for each type of adaptation.

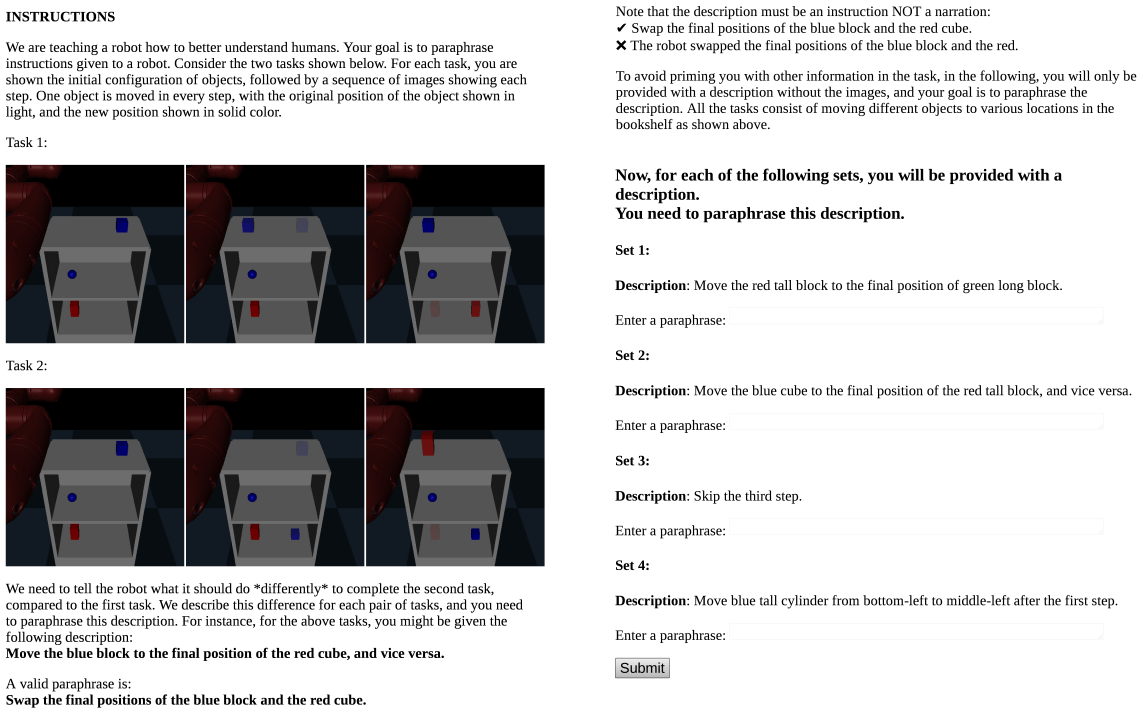


Figure 5.4: Interface for collecting paraphrases using Amazon Mechanical Turk

Table 5.1: Types of adaptations used in our experiments. For each type, an example of source and target tasks is shown.

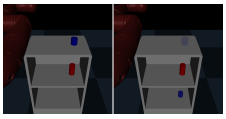
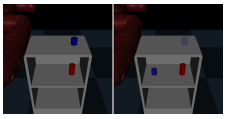
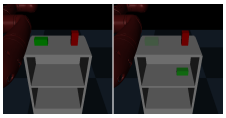
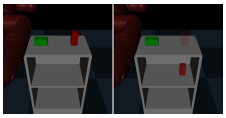
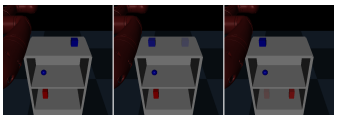
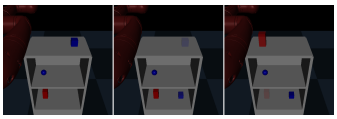
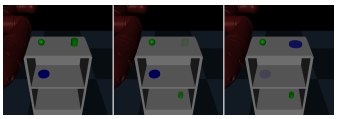
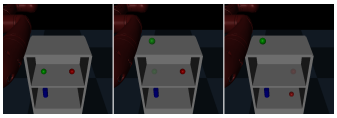
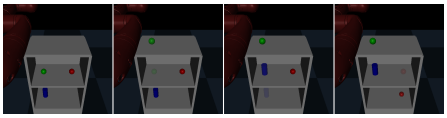
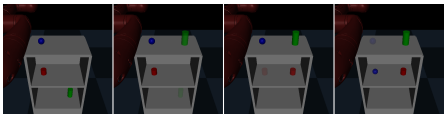
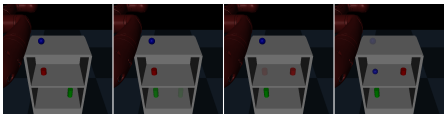
1.	**Same object, different place location**	
		
	Example Source Task	Example Target Task
2.	**Different object, same place location**	
		
	Example Source Task	Example Target Task
3.	**Move two objects, with swapped final locations**	
		
	Example Source Task	Example Target Task
4.	**Delete a step**	
		
	Example Source Task	Example Target Task
5.	**Insert a step**	
		
	Example Source Task	Example Target Task
6.	**Modify a step**	
		
	Example Source Task	Example Target Task

Table 5.2: Examples of template-generated and natural language descriptions collected using AMT.

	Template	**Natural language paraphrase**
1.	Move the cylinder to middle left.	Place the cylinder in the middle left
2.	Move the red tall block to the final position of green long block.	Place the red tall block in the green longs blocks final position
3.	Skip the third step.	do not do the third step
4.	In the first step, move the green tall cylinder from bottom right to bottom left.	for the first step, put the green tall cylinder in the bottom left position
5.	Move blue tall cylinder from bottom left to middle left after the first step.	For the second step move the blue tall cylinder to the middle left
6.	Move the blue cube to the final position of blue tall cylinder.	Swap the blue cube with the red cube on bottom shelf.
7.	Move blue tall block from top right to bottom left after the second step.	Move blue tall square from upper option to base left after the subsequent advance.

For each pair of source and target tasks in the dataset, we need a linguistic description of the difference between the tasks. We start by generating linguistic descriptions using a set of templates, such as, "Move `obj1` instead of `obj2` to the same position." We ensure that for most of these templates, the target task cannot be inferred from the description alone, and thus, the model must use both the demonstration of the source task and the linguistic description to infer the goal for the target task.

Next, we collected natural language for a subset of these synthetic (i.e. template-generated) descriptions using Amazon Mechanical Turk (AMT). Workers were provided with the template-generated descriptions, and were asked to paraphrase these descriptions. The interface used for collecting natural language descriptions using Amazon Mechanical Turk is shown in Figure 5.4.

We applied a basic filtering process to remove clearly bad paraphrases, such as those

with 2 words or less, and those that were identical to the given descriptions. We did not make any edits to the paraphrases, like correcting for grammatical or spelling errors. Some examples of template-generated and natural language descriptions obtained using AMT are shown in Table 5.2.

It can be observed that while the first four rows in the table are valid paraphrases, the remaining three paraphrases could be ambiguous depending on the source and target tasks. For instance, in row 5, the target task involves an *extra* step after the first step, while the natural language paraphrase could be interpreted as *modifying* the second step. In row 6, the natural language description is not a valid paraphrase, while in row 7, the paraphrase is difficult to comprehend. We manually analysed a small subset of the collected paraphrases, and found that about 15% of the annotations were ambiguous / non-informative. Some of this noise could be addressed by modifying the data-collection pipeline, for example, by providing more information about the source and target tasks to humans, and filtering out non-informative / difficult to comprehend paraphrases.

A vocabulary was created using the training split of the synthetic and natural language descriptions, discarding words that occurred fewer than 10 times in the corpus. While encoding a description using the resulting vocabulary, out-of-vocabulary tokens were represented using the `<unk>` symbol.

5.5 Experiments

Dataset. For each adaptation, 6,600 pairs of source and target tasks were generated along with the template-based descriptions. Of these, 600 templates were used for each adaptation to collect natural language descriptions using Amazon Mechanical Turk. Thus, our dataset

consisted of 6,600 examples in total for each adaptation, of which 6,000 examples consisted of synthetic language, and 600 consisted of natural language. Of the 6,000 synthetic examples per adaptation, 5,000 were used for training, 500 for validation, and the remaining 500 for testing. Similarly, of the 600 natural language examples per adaptation, 500 were used for training, 50 for validation, and 50 for testing. This gave us a training dataset with 33,000 examples, and validation and test datasets with 3,300 examples each, across all adaptation types.

Evaluation Metrics. For each experiment, the trained model predicts the reward or value of the given state s. When using the value function, the trained network is used to predict the values for all states $s \in S_{tgt}$. When using the reward function, the trained network is used to predict the rewards for all states $s \in S_{tgt}$, from which the optimal value function is computed using dynamic programming. In both the cases, if the state with the maximum value matches the goal state for the target task, g_{tgt}, the task is deemed to be successful, since we can define a policy that plans to the state with the maximum predicted value to solve the task. We train the models using the entire training set (i.e. both synthetic and natural language examples across all adaptations), and report the percentage of successfully completed target tasks for both synthetic and natural language descriptions. For each experiment, we tune the hyperparameters on the validation set, and report the success rate on the test set corresponding to the setting yielding the maximum success rate on the validation set.

Table 5.3: Success rates of different models

	Experiment	Success rate (%)	
		Synthetic	**Natural**
1.	LARVA; reward prediction	97.8	75.7
2.	LARVA; value prediction	97.7	73.3
3.	LARVA; no target goal supervision	20.0	2.7
4.	LARVA; no language	20.7	22.3
5.	LARVA; no source demonstration	4.2	3.3
6.	NN without decomposition	1.8	1.0
7.	LARVA: Compostionality – red box	87.6	62.4
8.	LARVA: Compostionality – blue cylinder	89.4	65.9

Results

In this section, we describe the performance of our full model, along with various ablations. Our results are summarized in Table 5.3.

First, we evaluate our full LARVA model, with both reward and value predictions (rows 1 and 2 in Table 5.3). In both cases, the models result in successful completion of the target task more than 97% of the time with synthetic language, and more than 73% of the time with natural language. The drop in performance when using natural language can partly be attributed to the 15% of paraphrases that are potentially ambiguous or uninformative, as discussed in Section 5.4.2, while the remaining performance gap is likely because natural language has more variation than synthetic language. Better data collection and more complex models could be explored to bridge this gap further.

The similar performance when predicting rewards and values is not unexpected—we observed in our experiments that training the target goal prediction module was more challenging than training the the reward or value networks. Since the target goal prediction

module is identical for both reward and value predictions, the performance in both cases is upper-bounded by the performance of the target goal prediction module. For domains with complex dynamics, reward and value prediction might result in significantly different success rates.

5.5.1 Ablations

Next, we discuss ablations of our model. We present the results only with value prediction, since as noted, both reward and value predictions perform similarly.

1. To study the effect of target goal supervision for training the target goal predictor, we remove $\mathcal{L}_{goal}$, optimizing the network using the ground-truth values only. Row 3 in Table 5.3 shows that this drastically degrades performance, confirming the efficacy of target goal supervision.

2. To ensure that most tasks require using information from both the source demonstration and the natural language description, we run unimodal baselines, wherein the network is provided with only the source demonstration (row 4) or only the language (row 5). As expected, both the settings result in a substantial drop in performance. Interestingly, using only the source demonstration results in over 20% successful completions. This is because given the set of adaptations, the source demonstration constrains the space of target configurations more effectively (e.g. if the source task consists of three steps, the target task must contain at least two of those steps, since source and target tasks differ by only one step for all adaptations).

3. Finally, we experiment with an alternate neural network architecture, that does not

decompose the learning problem into target goal prediction and value prediction. The source demonstration, the language, and the target state s are all encoded independently, and concatenated, from which the value for state s is predicted. Row 6 shows that the resulting model achieves a very low success on target tasks, demonstrating the importance of decomposing the learning problem as in LARVA.

5.5.2 Compositionality

In the experiments so far, the data were randomly partitioned into training, validation, and test splits. However, a key aspect of using language is the ability to *compose* concepts. For instance, humans can learn concepts like "blue box" and "red cylinder" from independent examples, and can recognize a "blue cylinder" by composing these concepts without ever having seen examples of the new concept.

To evaluate whether our proposed model can exhibit the ability to compose concepts seen during training, we create 2 new splits of our data—in both the splits, the training data consists of all datapoints that do not contain any blue cylinders or red boxes. In the first split, the validation set consists of all datapoints containing blue cylinders, while the test set consists of all datapoints containing red boxes. In the second split, the validation and test sets are swapped.*

We train LARVA on these new splits (using value prediction), and report the success rate on the test set in Table 5.3, rows 7 and 8. As can be observed, our model is able to successfully complete a large fraction of the target tasks, by composing concepts seen during

*Datapoints containing both a blue cylinder and a red box are discarded.

Table 5.4: Success rates (%) when using varying amounts of synthetic and natural language data to train LARVA. The row labels show the number of natural language examples used while the column labels show the number of synthetic language examples used.

(a) Synthetic language test set

# natural	# synthetic			
	0	**7,500**	**15,000**	**30,000**
0	-	83.8	93.2	97.3
750	3.0	85.8	93.4	97.2
1,500	5.9	85.8	91.9	96.8
3,000	30.1	88.1	93.6	97.7

(b) Natural language test set

# natural	# synthetic			
	0	**7,500**	**15,000**	**30,000**
0	-	48.7	46.3	51.0
750	1.7	60.7	58.7	65.7
1,500	4.3	63.0	64.0	73.3
3,000	29.3	68.7	72.0	73.3

training, however, there is room for further improvement by using richer models.

5.5.3 Amount of Data Needed

In order to better understand the amount of data needed, we trained LARVA with varying amounts of synthetic and natural language training examples (using value prediction). The results are summarized in Table 5.4.

Unsurprisingly, on the synthetic language test set, the number of natural language examples only makes a difference when the number of synthetic language examples in the training set is small. The results on the natural language test set are more informative. In particular, if no natural language examples are used for training, the model is only able to successfully complete about 50% of the tasks, even as the amount of synthetic language data is increased. Furthermore, using 1,500 natural language examples instead of 3,000 with 30,000 synthetic language examples results in a comparable performance as the full set. Similarly, halving the amount of synthetic language data (i.e. 15,000 examples instead of 30,000) when using the full natural language set results in only a small reduction in

performance.

These results suggest that using additional synthetic language or natural language data compared to our full set will likely not result in a significant performance improvement, and thus, improving the performance when using natural language requires filtering out low quality natural language data, and using richer models. Also, it is clear that some amount of natural language training data is needed to successfully generalize to natural language test cases.

5.6 Conclusions

We proposed a novel setting for zero-shot task adaptation—given a demonstration of a source task, and a natural language description of the difference between the source and target tasks, the agent needs to complete the target tasks without any demonstrations. This can be effective in teaching multiple related tasks to an agent by providing one (or a few) demonstration(s), and multiple low-effort linguistic descriptions. We introduced LARVA, a framework that decomposes the problem of learning the reward/value function for the target task into two subproblems: (1) inferring the goal state of the target task, and (2) predicting the reward/value function for a state under the target task, given the inferred goal state. Our experiments on the Organizer environment show that the method achieves more than 95% success on template-based descriptions, and about 75% success on natural language descriptions. Further, our model is able to compose concepts seen during training to reason about unseen combinations at test time, and our experiments with varying amounts of training data used show that some amount of natural language training data is needed to perform well on natural language test cases.

Chapter 6

Relational Language-guided Task Adaptation for Imitation Learning

6.1 Introduction

In Chapter 5, we introduced a novel problem setting for zero-shot task adaptation using natural language, wherein the agent needs to learn a *target* task, given a demonstration of a slightly different *source* task and a natural language description of the source and target tasks. We presented an approach that predicts the goal image for the target task, and consequently learns a reward/value function. In this chapter, we extend the approach, which we refer to as **relational reward adaptation**, to address some of its limitations.

First, the prior approach did not explicitly reason about the structure of the tasks or the environment. For instance, consider the scenario shown in Figure 6.1, which requires reasoning about objects and their relationships. In this work, we represent states as a set of *entities* and present a transformer-based model that reasons about the relationships of these entities with each other and with the provided linguistic description, to infer the reward function for the target task.

Second, in prior approach, the evaluation protocol did not have a policy learning phase, and only involved checking if the inferred reward/value function was maximized at the true goal state for the target task. This metric is not applicable to continuous control tasks,

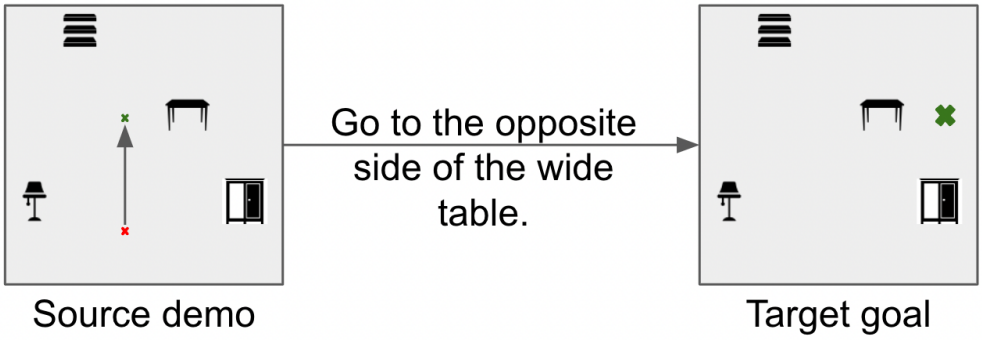


Figure 6.1: Example of the setting: The left image shows a demonstration of the source task, where the red point is the initial location of the agent, and the green point is the final location. The image on the right shows the target task, with the desired goal location marked with the green 'x'. The objective is to train an agent to perform the target task without any demonstrations of the target task, which requires reasoning about relative positions of entities.

since the maximum of the inferred reward/value function will often not coincide exactly with the true goal state. Further, it does not test whether the inferred reward/value function is useful for learning a policy using RL. In this work, we incorporate an explicit policy learning phase to address this limitation in evaluation.

Finally, the prior approach assumed access to data of the form (source demonstration, language, target reward/value function) for the supervised learning phase, whereas the new approach learns from data of the form (source demonstration, language, target demonstration). Having the ground truth reward/value function for the target task as in the prior approach may be more desirable if defining these reward/value functions for various target tasks in the data is relatively easy, since rewards or values are unambiguous compared to demonstrations. However, in many domains, it may be easier to implement a planner from which generating demonstrations is more convenient, in which case the setting we present in

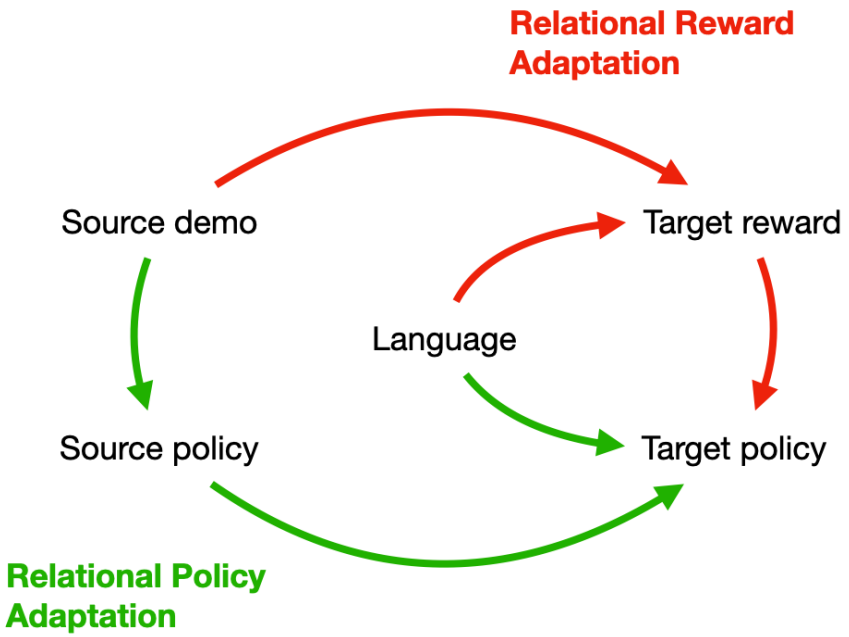


Figure 6.2: We present two independent approaches to learn a target task policy – relational reward adaptation (Section 6.3) and relational policy adaptation (Section 6.5). In Section 6.8, we show how to combine these two approaches.

this chapter would be more suitable. Further, the symmetry between source and target data in the new approach would allow using existing multitask datasets with trajectories, where similar tasks may be annotated with linguistic descriptions of the differences.

We also develop an alternate approach, wherein, instead of inferring the goal state for the target task, the policy for the source task is adapted to produce a policy for the target task. We refer to this approach as **relational policy adaptation**. This requires that the transition dynamics of the source and the target tasks be identical, which is true in a lot of real-world scenarios.

We show that our approaches – relational reward adaptation and relational policy

adaptation – can be combined to build a unified system for task adaptation. A schematic diagram of the overall framework is shown in Figure 6.2.

The setting and the approaches we develop in this chapter are related to transfer learning and relational reasoning, which were discussed in more detail in Section 2.1.3.

We create two new benchmarks that require reasoning about the relative positions of entities, which we describe next, that are used to evaluate the proposed reward adaptation and policy adaptation approaches.

6.2 Benchmark Datasets

We created two benchmark environments: Room Rearrangement and Room Navigation. For each environment, we construct a dataset of (source demonstration, language, target demonstration) triplets, where the demonstrations are generated using a planner, and the language is generated using templates. Further, we collect natural language descriptions for a subset of these datapoints, using Amazon Mechanical Turk (AMT). The details of the environments, the datasets, and natural language data collection are described below.

Objects. We use a common set of objects in both the environments, which we describe here. There are 6 distinct nouns—`Chair`, `Table`, `Sofa`, `Light`, `Shelf`, and `Wardrobe`. Further, each object can have one of 6 attributes—`Large`, `Wide`, `Wooden`, `Metallic`, `Corner`, and `Foldable`. The resulting 36 (attribute, noun) pairs are split into 24 train pairs, 6 validation, and 6 test pairs. Datapoints for each split use pairs only for that split. This ensures that the model will encounter unseen (attribute, noun) pairs during test time. The agent is also treated as an entity with both the attribute and the noun set to a special symbol `Agent`.

Room Rearrangement Environment. The Room Rearrangement Environment consists of a 5×5 grid, with 2 distinct objects. The objective is to move each object to a desired goal position. The agent and the objects are spawned randomly in the grid. The action space for the agent consists of 7 actions—`Up`, `Down`, `Left`, `Right`, `Grasp`, `Release`, and `Stop`. If the agent is on a cell that contains another object, the `Grasp` action picks up the object, otherwise it leads to no change. A grasped object moves with the agent, until the `Release` action is executed. The `Up`, `Down`, `Left`, and `Right` actions move the agent (and the grasped object, if any) by one unit in the corresponding direction, except when the action would result in the agent going outside the grid, or the two objects on the same grid cell. In these cases, the action doesn't result in any change. The `Stop` action terminates the episode.

Room Navigation Environment. The Room Navigation Environment consists of a 2D arena, $(x, y) \in [-100, 100]^2$, with 4 distinct objects. The agent is spawned at a random location in the arena, and needs to navigate to a desired goal position. The action space for the agent is $(dx, dy) \in [-1, 1]^2$. The episode terminates when the agent takes an action with an ℓ_2-norm less than 0.1.

Adaptations. For each domain, we create three types of adaptations. For Room Rearrangement, these adaptations involve specifying an absolute change in the goal position of each entity, the relative change in the goal position of one entity with respect to the other, and swapping the goal positions of the entities. For Room Navigation, these adaptations involve moving closer to an entity, moving further away from an entity, and going to the opposite side of an entity. See Figure 6.3 for examples of these adaptations.

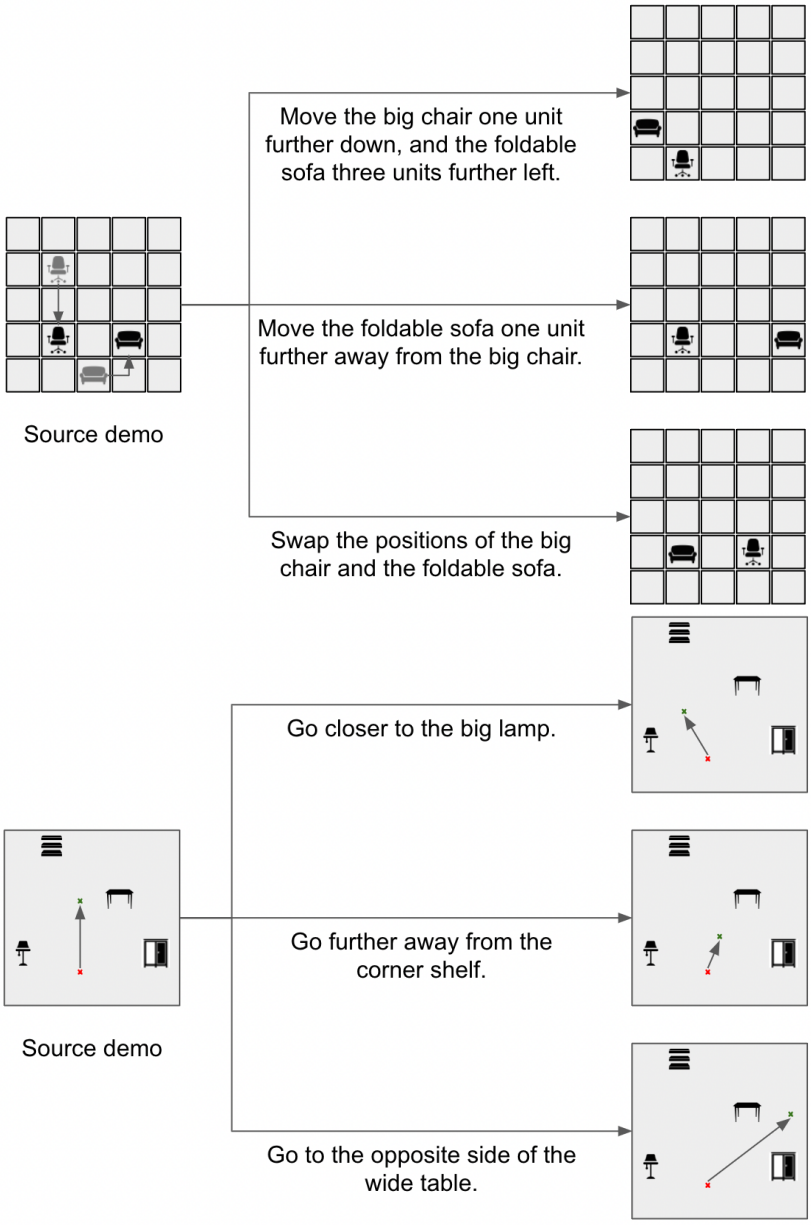


Figure 6.3: Adaptations used in the Room Rearrangement (top) and Room Navigation (bottom) domains.

Together, these environments cover various types of adaptations, such as specifying modifications to one versus several entities, providing absolute modifications to an entity's position (e.g., "move the table one unit further left") versus modifications that are relative to other entities (e.g., "move the table one unit away from the sofa"). Further, these domains cover different types of MDPs, with Room Rearrangement being a discrete state and action space environment, with a relatively short horizon, while Room Navigation being a continuous state and action space environment, with a longer horizon. (On average, an optimal policy completes a task in the Room Rearrangement domain in about 30 steps, while in the Room Navigation domain in about 150 steps.) Finally, the Room Navigation domain has a unique optimal path (i.e. a straight line path between the initial state and the goal state), while the Room Rearrangement domain admits multiple optimal paths (e.g. if reaching an entity requires taking 2 steps to the right and 1 step upwards, these steps can be performed in any order). Thus, these two domains make a robust testbed for developing techniques for the proposed problem setting.

Language Data. For each pair of source and target tasks in the dataset, we need a linguistic description of the difference between the tasks, which is an instance of one of the six adaptations shown in Figure 6.3. We start by generating these descriptions using a set of templates, such as, "Move `attribute1 obj1` one unit closer to the `attribute2 obj2`". We ensure that for all these templates, the target task cannot be inferred from the description alone, and thus, the model must use both the demonstration of the source task and the linguistic description to infer the goal for the target task. Next, we crowdsourced natural language for a subset of these synthetic (i.e. template-generated) descriptions using AMT.

Table 6.1: Examples of template-generated and natural language descriptions collected using AMT.

	Template	**Natural language paraphrase**
1.	go further away from the metallic table	Increase your distance from the metallic table.
2.	go closer to the foldable light	Move in the direction of the light that is foldable
3.	go to the opposite side of the corner light	Move across from the corner light.
4.	move the large chair one unit farther from the wide couch	Increment the distance of the big chair from the wide couch by one.
5.	move corner table two units further left and metallic shelf one unit further backward	slide the corner table two units left and move the metal shelf a single unit back
6.	move the large table to where the large sofa was moved, and vice versa	swap the place of the table with the sofa

Workers were provided with the synthetic descriptions, and were asked to paraphrase these descriptions, using a setup similar to the one used in Chapter 5. See Table 6.1 for some examples of template-generated and natural language descriptions obtained using AMT.

Dataset. For each adaptation template, 5,000 datapoints were generated for training, 100 for validation of the reward and goal learning, 5 for tuning the RL hyperparameters, and 10 for the RL test set. This gave us (1) a training dataset with 15,000 datapoints for each benchmark, (2) a validation dataset for supervised learning with 300 datapoints, (3) a validation set for RL with 15 datapoints, and (4) a test set for RL with 30 datapoints. We collect natural language paraphrases for 10% of the training datapoints, and all the datapoints in the other splits.

6.3 Relational Reward Adaptation: Approach

We propose a framework that takes in a source demonstration, τ_{src}, and the difference between the source and target tasks described using natural language, l, to output the reward function for the target task R_{tgt}. This reward function is then used to learn a policy for the target task using reinforcement learning (RL).

We assume access to a training set $\mathcal{D} = \{(\tau^i_{src}, \tau^i_{tgt}, l^i)\}_{i=1}^N$, where τ^i_{src} is a demonstration for the source task of the i^{th} datapoint, τ^i_{tgt} is a demonstration for the target task of the i^{th} datapoint, and l^i is the natural language description of the difference between the source task and the target task for the i^{th} datapoint.

We propose a relational model since many adaptations require reasoning about the relation between entities (e.g. "Move the big table two units away from the wooden chair"). Since entity extraction is not the focus of this work, we assume access to a set of entities for each task, where each entity is represented using two one-hot vectors, corresponding to an attribute and a noun. The details of attributes and nouns used in our experiments is given in Section 6.2. Further, each state is represented as a list, where element i corresponds to the (x, y) coordinates of the i^{th} entity. Finally, we assume that the number of entities, denoted as N_{entities}, is fixed for a given domain.

We define the reward function $R(s, s')$ using a potential function as, $R(s, s') = \phi(s') - \phi(s)$. Thus, the problem of reward learning is reduced to the problem of learning the potential function $\phi(s)$. We decompose the potential function learning problem into two subproblems: (1) predicting the goal state for the target task given the source goal and the language, $g_{tgt} = \mathit{Adapt}(g_{src}, l)$, and (2) learning a distance function between two states, $d(s, s')$. The

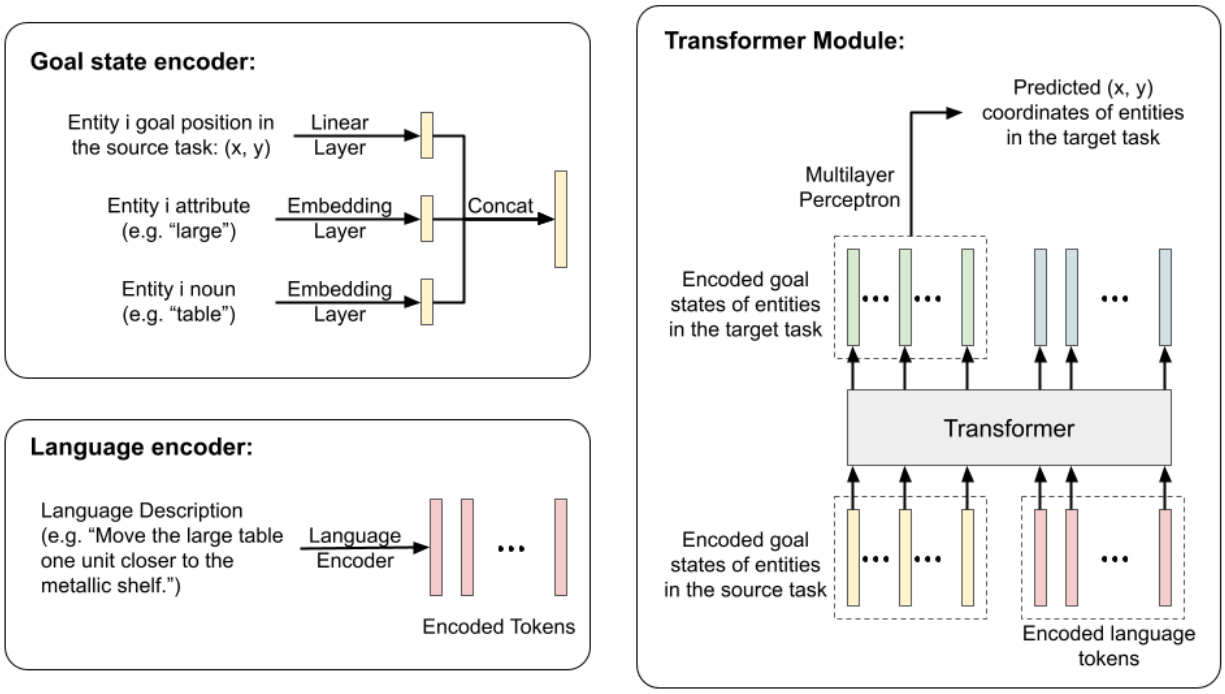


Figure 6.4: Neural Network architecture for relational goal prediction.

potential function for the target task is then defined as $\phi_{tgt}(s|g_{src}, l) = -d(s, Adapt(g_{src}, l))$.

6.3.1 Goal Prediction

Given a set of entities E, a goal state for the source task represented as a list of positions of each entity (g_{src}), and a natural language description of the difference between the tasks (l), the goal predictor network is trained to predict the goal state for each entity in the target task (g_{tgt}).

Entity Encoder. We assume each entity is represented using two one-hot vectors, corresponding to an attribute and a noun. These two encodings are passed through embedding

layers for attributes and nouns respectively, to obtain dense vector representations, which are then concatenated to get the final vector representation of the entity, $\mathbf{e_i}$.

Position Encoding. As mentioned earlier, a state is represented as a list of entity positions. Each entity position is projected to an embedding space using a linear layer. This position encoding is concatenated with the final vector representation of the entity $\mathbf{e_i}$. Similarly, the goal state of each entity in the source task is projected to the embedding space using the same linear layer, followed by concatenation with the vector representation of the entity.

Language Encoders. We experiment with 4 different ways of encoding language. First, we use a pretrained CLIP model [Radford et al., 2021], which has been shown to be effective at language grounding tasks, to obtain an embedding for each token in the description. The parameters of the pretrained model are kept frozen during the training of the downstream network. Second, instead of a pretrained CLIP model, we use a pretrained BERT model (base, uncased; [Devlin et al., 2018]). As before, the pretrained model is kept frozen. Third, instead of using a pretrained BERT model, we experiment with a randomly initialized BERT model that is learned along with the downstream network. Finally, we use GloVe word embeddings [Pennington et al., 2014] followed by a two-layer bidirectional LSTM [Hochreiter and Schmidhuber, 1997]. The GloVe+LSTM and randomly initialized BERT models are more flexible, allowing them to learn representations for words and sentences that are specialized for the task at hand, while the pretrained CLIP and BERT models can potentially leverage the external knowledge seen during the pretraining phase. While pretrained BERT

encodes language independent of its grounding, the CLIP model is pretrained on multimodal data, which is likely more useful for our setting which requires language grounding.

Transformer-based Attention Module. Since the domains we consider in this work involve relational adaptations that involve reasoning about relative positions of entities, as well as how they interact with language, we use a transformer-based model to learn these relations. The encoded goal states for the source task, the current positions of all the entities, and the token embeddings generated by the language encoder are concatenated to create a single sequence of tokens, which are passed through a transformer layer. The first N_{entities} tokens of the output sequence are projected back to the $\mathbb{R}^2$ space using a multi-layer perceptron with three linear layers and ReLU non-linearities between them to get the predicted goal state of each of the N_{entities} entities under the target task.

A diagram of the goal predictor neural network is shown in Figure 6.4.

6.3.2 Distance Function Learning

The distance function takes in two states s and s', and predicts the distance between them. While an ℓ_2-distance between the states can be directly computed, this may not be ideal in many domains. Therefore, we learn a neural network ψ with two linear layers and a ReLU non-linearity between them to project the raw states into an embedding space. The distance between the states s and s' is then computed as $d(s, s') = \|\psi(s) - \psi(s')\|_2$.

6.3.3 Training

To train the model, we assume access to a dataset $\mathcal{D} = \{(\tau^i_{src}, \tau^i_{tgt}, l^i)\}_{i=1}^N$.

Goal Prediction. The goal prediction module is trained by using the final states in the source and target demonstrations, as the source and target goals respectively. We minimize the mean absolute error between the gold target goal state, g_{tgt} and the predicted target goal state, $\hat{g}_{tgt}$:

$$L_{goal} = \frac{1}{N} \sum_{i=1}^{N} \|g_{tgt} - \hat{g}_{tgt}\|_1$$

Distance Function. To train the distance function, two states s_i and s_j are sampled from a demonstration τ, which can be the source or the target demonstration for the task, such that $i < j$. The model is trained to predict distances such that $d(g, s_i) > d(g, s_j)$, where g is the goal state for the demonstration. This is achieved using the following loss function:

$$L_{dist} = -\sum_{s_i, s_j, g} \log \left(\frac{\exp(d(g, s_i))}{\exp(d(g, s_i)) + \exp(d(g, s_j))} \right)$$

This loss function has been shown to be effective at learning functions that satisfy pairwise inequality constraints [Christiano et al., 2017, Brown et al., 2019].

Optimization. The goal prediction and distance function modules are independently trained using the dataset $\mathcal{D}$. We used an Adam optimizer [Kingma and Ba, 2014] to train the networks for 100 epochs each. A validation set was used to tune hyperparameters via random search.

The learned goal prediction and distance function modules are combined to obtain a reward function for the target task, which is then used to train a policy using reinforcement learning. More details about this step are provided in Section 6.4.

6.4 Relational Reward Adaptation: Experiments

6.4.1 Details about the setup

Policy Training. The goal prediction and distance function learned on the training set are used to train a policy for each task in the test set with the hyperparameters found to be optimal for the 5 validation RL tasks. We report the total number of successful episodes at the end of 500,000 and 100,000 timesteps respectively for these domains, averaged over three RL training runs per target task. We use PPO as the RL algorithm for all our experiments [Schulman et al., 2017, Raffin et al., 2021]. For the Room Rearrangement domain, the agent and the entities are initialized uniformly at random anywhere on the grid at the beginning of each episode. For the Room Navigation domain, the entities are initialized in the same positions as in the source task, while the agent is initialized uniformly at random in the arena.

Evaluation Metrics. In the Room Rearrangement domain, an episode is deemed successful if both the entities are in the desired goal locations when the agent executes `Stop`, while for the Room Navigation domain, an episode is deemed successful if the ℓ_2-distance between the agent's final position and the desired goal position is less than 5 units. (Recall that the total arena size is 200×200 units, and the episode ends when the agent executes an action with an ℓ_2-norm less than 0.1 unit.)

6.4.2 Results

In this section, we describe the performance of our full model, along with various ablations. Our results are summarized in Table 6.2. Unless stated otherwise, we use the full

Table 6.2: Success rates for different models on Room Rearrangement and Room Navigation domains. We report both the raw success rates (unnormalized), and success rates normalized by the oracle setting performance.

Setting	**No. of successes**			
	Rearrangement		**Navigation**	
	Unnormalized	**Normalized**	**Unnormalized**	**Normalized**
Reward Adaptation	2996.02 ± 136.21	68.05 ± 3.09	247.98 ± 20.51	73.54 ± 6.08
Oracle	4402.78 ± 410.67	100.00 ± 9.33	337.22 ± 7.34	100.00 ± 2.18
Zero reward	121.02 ± 4.25	2.75 ± 0.10	0.29 ± 0.04	0.09 ± 0.01
True goal, predicted distance	4164.80 ± 337.83	94.59 ± 7.67	362.13 ± 12.18	107.39 ± 3.61
Predicted goal, true distance	3706.80 ± 200.46	84.19 ± 4.55	196.49 ± 12.97	58.27 ± 3.85
Synthetic language	3827.64 ± 141.79	86.94 ± 3.22	317.11 ± 49.26	94.04 ± 14.61
Non-relational goal prediction	869.89 ± 115.12	19.76 ± 2.61	0.38 ± 0.17	0.11 ± 0.05

set of synthetic and natural language descriptions for supervised training, and only natural language descriptions during testing for RL.

The first row corresponds to our full reward adaptation model, that learns a relational goal prediction model, and a distance function, which are then combined to form a reward function for RL. The target tasks are trained with natural language descriptions collected using AMT. The next two rows serve as approximate upper and lower bounds respectively. The second row corresponds to an oracle setting, wherein, a policy is trained with the true goal state for the target task, and the potential of a state is defined as the distance between the current state and the goal state. We use the ℓ_1-distance for Rearrangement, and the ℓ_2-distance for Navigation. For the third row, we define the reward function to be uniformly zero, and this result tells us how well a random policy would do on our target tasks.

We can observe that the proposed model is substantially better than the lower bound,

but is about 70% as good as the oracle. As such, there is quite a bit of room for improvement to achieve performance close to the oracle.

Next, we experimented with different ablations of our full reward adaptation model. Since our full model consists of two learned components, the goal prediction module, and the distance function, we first study the impact of each of these components independently. We experiment with the following two settings: (1) the true target goal state, with the learned distance function (Row 4), and (2) the learned target goal prediction, with the true distance function (Row 5). As expected, the distance function is easy to learn in these domains, and using the learned distance function instead of the true distance function leads to a small or no drop in performance. Most the performance drop comes from the goal prediction module, and therefore future modeling innovations should focus on improving the goal prediction module.

Next, we look at the performance difference between synthetic and natural language. Row 6 in Table 6.2 shows the number of successful episodes when using synthetic language only, both during training the goal prediction model, and for learning the target task policy using RL during testing. In both the domains, using synthetic language is significantly better than using natural language, and is comparable to the oracle.

In order to analyze the benefit of using the relational model, we compare our approach against a non-relational model. Row 7 shows the results when using a non-relational model, where we use a multilayered perceptron with three linear layers, that takes in the entity vectors, goal positions of all entities in the source task, and the CLIP embedding of the final token in the description, all concatenated together as a single input vector, and outputs the goal positions of all entities in the target task as a single vector. This model is significantly

worse than the relational model on both the domains, highlighting the benefit of using a relational approach for these tasks.

Finally, we explore whether language descriptions that are opposite in meaning lead to opposite modifications to the predicted goal state. That is, if the language description "Go closer to the wide table" leads to a change between the source and target goal denoted by Δ, then, does the language description "Go further away from the wide table" lead to a $-\Delta$ change? We study this in the Navigation domain with synthetic language. For the adaptations involving moving closer to or away from an object, we replace the original language description with the opposite language description, and predict the goal state, which we denote as $g_{tgt}^{inverse}$. Thus, the model predicts that the modified language amounts to the change in goal state that can be computed as $\Delta = g_{tgt}^{inverse} - g_{src}$. The goal state for the original language description is then given by $g_{tgt} = g_{src} - \Delta = g_{src} - (g_{tgt}^{inverse} - g_{src})$. We use this goal state for training a policy on the test tasks, and get 301.00 ± 79.25 successes across all tasks, which is comparable to using the original language (Row 6 in Table 6.2). This suggests that the approach can be used to make predictions on concepts not seen during training, if the model has been trained on inverse language.

Of the language encoders we used, we did not find any substantial difference between different encoders. This is likely because the language descriptions used in our experiments were relatively simple.

6.5 Relational Policy Adaptation: Approach

Instead of learning a model to infer the reward function for the target task from the source demonstration and language, in this section, we explore an alternate approach

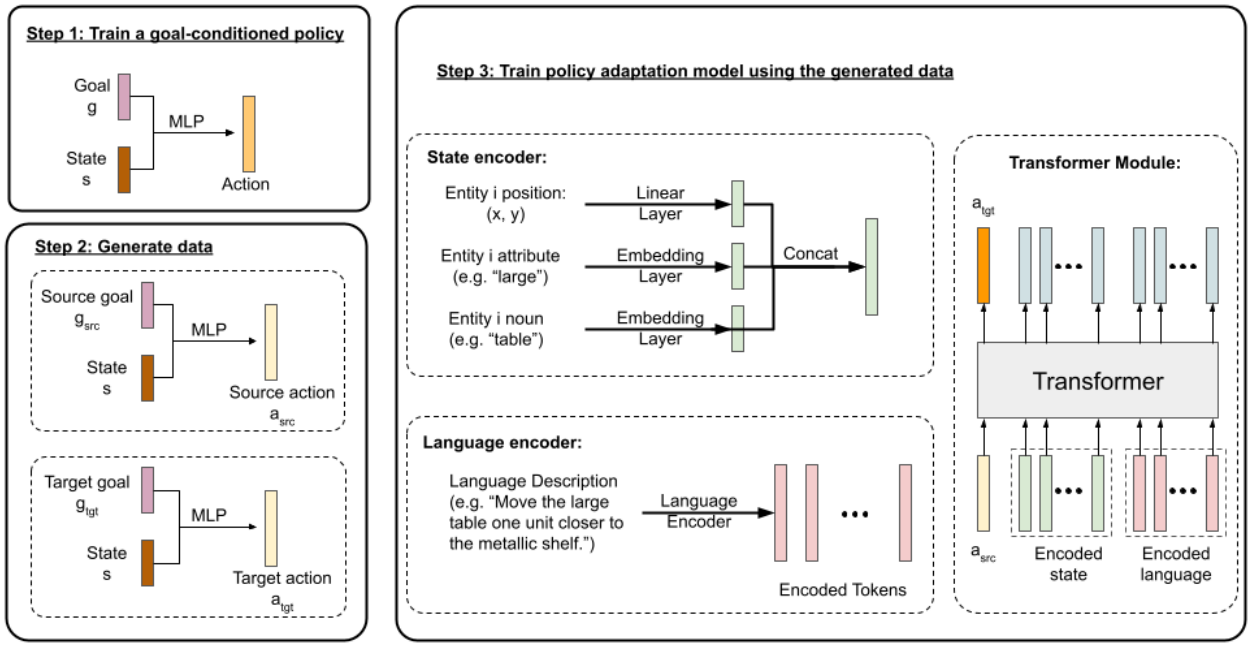


Figure 6.5: Policy Adaptation Approach

wherein we learn a model to infer the target task policy from the source task policy.

First, a goal-conditioned policy $\pi(s|g)$ is learned using all the source and target demonstrations—given the goal state for a task, g, (which is assumed to be the last state in the demonstration), and another state, s, we use behavior cloning to learn a policy that predicts the action to be taken at state s. We use a neural network to parameterize this policy, wherein the states g and s are concatenated and then passed through a multi-layer perceptron to predict the action at state s.

The learned model is then used to generate data of the form (state, language, source action, target action). For each datapoint of the form $(\tau^i_{src}, \tau^i_{tgt}, l^i)$, the states in the source and target demonstrations are passed through the learned goal-conditioned policy, passing

in the source task goal and the target task goal to obtain the actions in the source and target tasks respectively:

$$a_{src} = \pi(s|g_{src}) \ ; \quad a_{tgt} = \pi(s|g_{tgt})$$

This data is used to train a transformer-based adaptation model, that takes in the source action, the entities in the state s, and the language to predict the target action. The entities and language are encoded as in Section 6.3. See Figure 6.5 for a diagram of the approach.

During evaluation, we are given the source demonstration and language, as before. We use the goal-conditioned policy $\pi(s|g)$ to first predict the action for the current state under the source task, and then pass this predicted action, along with the entities and language to the adaptation model, to obtain the action under the target task. This action is then executed in the environment. The process is repeated until the `STOP` action is executed (or the maximum episode length is reached).

Note that this approach does not involve reinforcement learning to learn the policy.

6.6 Relational Policy Adaptation: Experiments

Here, we present the experimental setup and the results for the relational policy adaptation approach described above. First, the source task goal state and the current state are passed through the goal-conditioned policy to get the action for the source task. This is then passed through the adaptation model, along with the current state, the goal state for the source task, and language to predict the action for the target task. The predicted action is executed in the environment.

Table 6.3: Success rates for relational policy adaptation on the Room Rearrangement and Room Navigation domains.

Setting	Success Rate (%)			
	Rearrangement		Navigation	
	Natural Language	Synthetic Language	Natural Language	Synthetic Language
Relational Policy Adaptation	15.33	29.13	3.87	21.71
Relational Policy Adaptation with Goal Prediction	55.80	83.73	6.47	26.93

To evaluate this approach, we generate 100 rollouts using the trained models for each test task, and compute the number of successful episodes. For each rollout, we randomize the initial state as in the policy training experiments in Section 6.4.

The results are reported in Table 6.3. Row 1 shows the success rates when the actions predicted by the approach are executed until the `Stop` action is predicted. We see that the approach completes the target tasks about 20% of the time, except when using natural language in the Navigation domain. Some analysis of the results suggested that the actions predicted by the learned models usually take the agent towards the goal, but the models are not as good at learning the stopping criterion. As such, we used the goal prediction model trained using the relational reward adaptation approach to predict the goal state, and predicted the `Stop` action when the agent was within a small threshold of the predicted goal. The results using this approach are reported in Row 2 of Table 6.3. We can see that this results in a substantial improvement, particularly in the Rearrangement domain.

Thus, the techniques developed in Sections 6.3 and 6.5 can be combined to complete a large fraction of the target tasks, without any finetuning on the target environment.

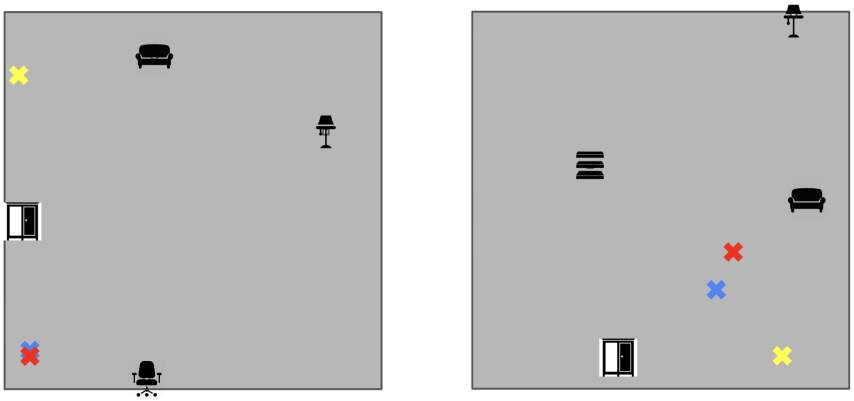


Figure 6.6: Visualization of predicted goal for two test datapoints. The yellow X denotes the goal position under the source task, and the red and blue X's denote the predicted and true goal positions under the target task.

6.7 Qualitative Analysis

In this section, we report some qualitative results on the Navigation domain with reward and policy adaptation approaches.

In Figure 6.6, we show two examples of goal prediction using the Relational Reward Adaptation approach. In the first example, the predicted goal state is quite close to the true goal state under the target task, suggesting that the model is able to successfully recover the target task. In the second example, the predicted goal is somewhat farther from the true goal. A plausible explanation is that the model was not able to disambiguate the entity being referred to by language, and therefore computes the target goal position as a linear combination of distances to multiple entities.

In Figure 6.7, we show three examples of paths followed by the agent when following the actions predicted by the Relational Policy Adaptation approach (without any finetuning).

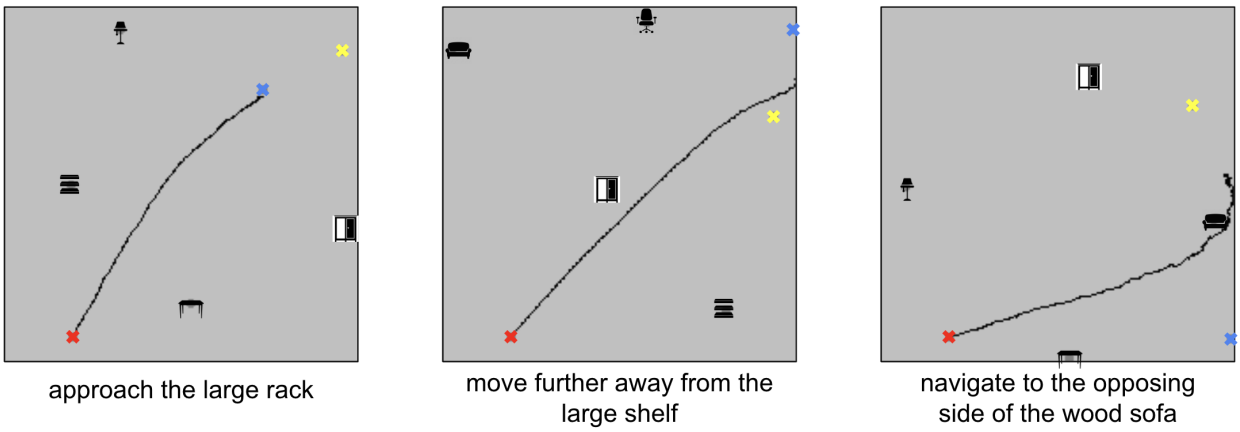


Figure 6.7: Visualization of predicted goal for two test datapoints. The red X denotes the initial position of the agent, the yellow X denotes the true goal position under the source task, and the blue X denotes the true goal position under the target task.

In the first example, we see that the agent successfully reaches and stops at the true goal position under the target task. In the other two examples, we see that the agent gets somewhat close to the goal position under the target task, but doesn't actually reach it (and is also going towards the goal position under the source task). The errors seem to get larger as agent gets closer to the target goal, motivating a modified training algorithm wherein datapoints could be weighted differently based on how close the agent is to the goal position. We leave this investigation for future work.

6.8 Combining Reward and Policy Adaptation

So far, we've described the relational reward adaptation approach that infers a reward function for the target task, and the relational policy adaptation approach that infers an initial policy for the target task. A natural question that arises is, can we combine these two approaches, for instance, by initializing the target task policy with the policy inferred by the

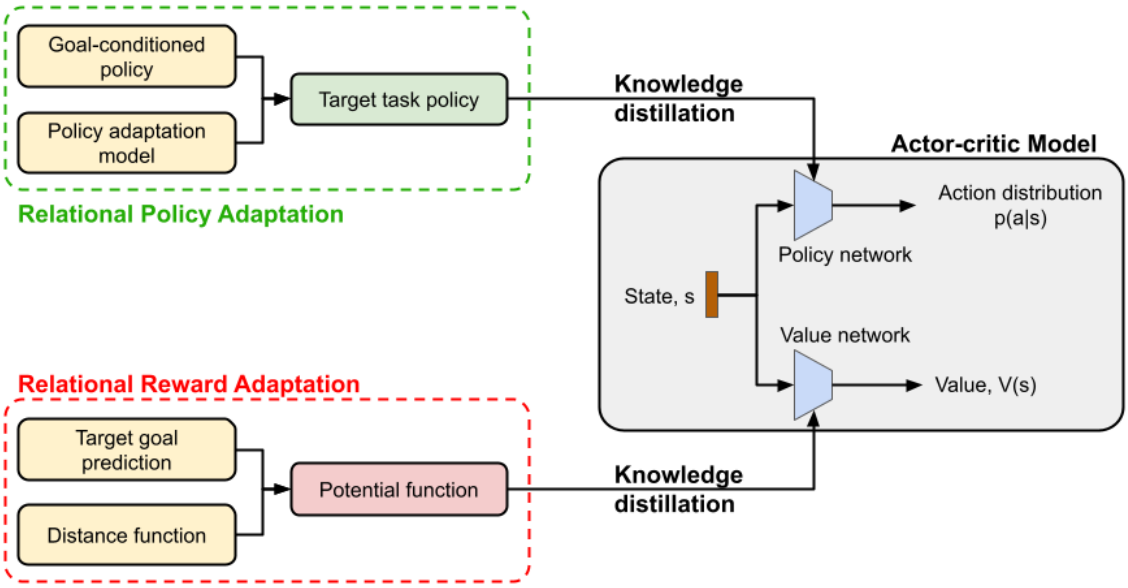


Figure 6.8: Initializing the value and policy networks of the actor-critic model using the reward adaptation and policy adaptation approaches.

relational policy adaptation approach, and then finetuning it with the rewards inferred by the reward adaptation approach using RL? In this section, we explore this idea further.

A straightforward approach is to use the adapted policy for initializing the action network in the actor-critic model trained using PPO, while still initializing the value network randomly. We experimented with this idea, and did not see any benefits of action network initialization suggesting that the value network must also be initialized in some way.

Interestingly, this can be achieved by using the potential function returned by the reward adaptation approach. Thus, we use the output of the policy adaptation approach to initialize the action network, and the output of the reward adaptation approach to initialize

the value network. The initialized networks are then finetuned using the reward function predicted by the reward adaptation approach.

Further, note that the architectures for the action and value networks in PPO are different from the architectures of the networks in policy and reward adaptations. Specifically, the networks in PPO are trained for a single target task, and therefore only take the state as input, whereas the policy and reward adaptation approaches are shared across different tasks and are therefore conditioned on language as well. As such, we cannot directly initialize network weights in PPO, and therefore use knowledge distillation [Hinton et al., 2015] to initialize the networks.

Our full approach is shown in Figure 6.8, and described below:

1. Train the reward adaptation and policy adaptation models using supervised learning independently, as detailed in Sections 6.3 and 6.5 respectively.

2. Use knowledge distillation to initialize the value network for PPO, where the loss function is the mean squared error between the value predicted by the PPO value network, and the potential predicted by the reward adaptation approach. Recall that the predicted potentials are obtained by combining the outputs of the target goal prediction and the learned distance function.

3. Use knowledge distillation to initialize the action network for PPO, where the loss function is the cross-entropy loss between the action probabilities predicted by the PPO action network, and the action predicted by the policy adaptation approach for the target task. Recall that the policy adaptation approach involves two steps

to predict the target task action: (1) predicting the action for the source task using the goal-conditioned policy, and (2) passing the predicted source task action to the adaptation model along with the goal state for the source task and the language to output the action for the target task.

4. Finetune the action and value networks using PPO with the rewards predicted by the reward adaptation approach.

For knowledge distillation, states from the demonstration data are sampled uniformly at random.

Importantly, we found that the action network initialized using knowledge distillation usually has a low entropy, and therefore finetuning it directly does not result in good performance. To ameliorate this issue, the entropy of the action network must be kept sufficiently high for it to still allow some exploration. In the continuous control case, we achieve this by increasing the standard deviation of the action network, tuned using the validation set. In the discrete domain, since there is no explicit parameter to control the entropy in the action network, the knowledge distillation step has an additional loss term to penalize low-entropy solutions.

For all the experiments here, we use synthetic and natural language for supervised learning, and only natural language for RL during evaluation. We report the number of successes when PPO is initialized randomly, as in Section 6.4, and when it is initialized using the adapted policy in Table 6.4. Further, Figure 6.9 shows the learning curves for these experiments.

Table 6.4: No. of successes when reward adaptation is combined with policy adaptation.

Setting	**No. of successes**	
	Rearrangement	**Navigation**
PPO without initialization	2996.02 ± 136.21	247.98 ± 20.51
PPO with initialization	8516.78 ± 894.35	430.80 ± 5.08

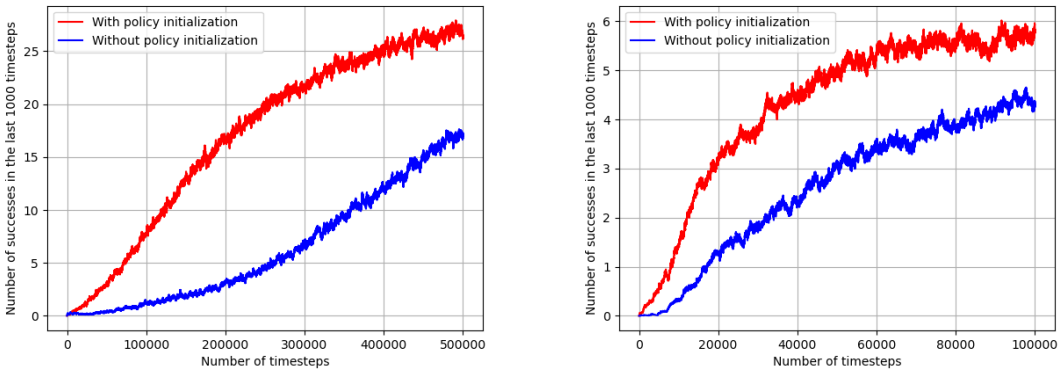


Figure 6.9: Learning curves comparing the policy training curves on target tasks when using uninitialized PPO networks and PPO networks initialized using policy adaptation, on the Rearrangement (left) and Navigation (right) domains.

We observe that on both the domains, initializing the policy network using the relational policy adaptation approach and the value network using the relational reward adaptation approach leads to a substantially faster policy learning on the target tasks, compared to randomly initialized PPO networks.

6.9 Conclusions

We extend the language-guided task adaptation problem introduced in Chapter 5 to relational adaptations, and introduce two new benchmarks—Room Rearrangement and Room Navigation. We presented two transformer-based relational approaches for these new

benchmarks. The first approach – relational reward adaptation – learns a transformer-based model that predicts the goal state for the target task, and learns a distance function between two states. These trained modules are then combined to obtain a reward function for the target task, which is used to learn a policy using RL. The second approach – relational policy adaptation – learns a transformer-based model that takes in a state, and the action at this state under the source task, to output the action at this state under the target task, conditioned on the source task goal and language. The data to train this model is generated by training a goal-conditioned policy on the demonstration data. Our experiments show that the proposed approaches can effectively be used to complete the target tasks on these new benchmarks, while there is still room for further improvement.

Chapter 7

Future Work

This dissertation sets up the groundwork for using natural language for task specification in RL and IL. We believe that there are several promising future directions worth exploring, which we discuss below.

7.1 Short-term Future Directions

Abstract Verbal Concepts. In Chapters 3, 4 and 5, the linguistic descriptions largely contained concepts that could be easily grounded to physical objects or verbs (e.g. "ladder", "toaster", "cylinder", "climb", "turn", "move"). In Chapter 6, we looked at some abstract relational concepts such as "closer", "away", and "opposite". However, there are other classes of abstract words that are worth experimenting with, such as those relating to the tempo of the motion (e.g. "slowly", "quickly", "gently"), or those that describe abstract properties of objects (e.g. "squishy", "heavy"), some of which would require grounding language to tactile sensors of the agent. Thomason et al. [2016] propose an approach to learn such concepts, which could be combined with the approaches presented in this dissertation.

Entity Modeling. The approach presented in Chapter 6 assumes we have ground-truth entities available. Future work could look at combining the approach with entity extraction

methods [Kosiorek et al., 2018]. Further, we assume a fixed set of entities across different tasks within a domain. Extending the approach to handle a variable set of entities is another avenue worth exploring, for instance using the approach presented in Zhou et al. [2022].

Lifelong Learning. The approaches proposed in this dissertation have a supervised learning phase before policy training, where language grounding is learned. This is followed by policy training where the model trained using supervised learning is queried, but not updated. This could be modified to get a lifelong learning system [Thrun, 1998], wherein the language grounding model is updated from the data collected during policy learning, so that it can perform better on future tasks. Further, Thomason et al. [2017] proposed an opportunistic active learning approach, wherein an agent learns to ground linguistic concepts opportunistically, which could then be used for future tasks. This framework could be combined with the approaches presented here, to build lifelong learning systems for task specification using natural language.

Additional Modalities. In this dissertation, we use natural language as a low-effort auxiliary modality to better communicate the user's intent to the learning agent. However, there are other modalities, such as gaze and facial expressions, that also convey information about the user's intent [Saran et al., 2020, Cui et al., 2020]. These modalities do not require any additional effort from the user, and could be combined with the approaches presented in this dissertation to further disambiguate the user's intent, especially for more complex tasks.

Feedback. The task adaptation framework presented in Chapters 5 and 6 is closely related to the problem of learning from feedback, wherein the agent's current suboptimal behavior

can be treated as the "source demonstration", and natural language could be used to describe how it should modify the behavior towards the target task. As such, it may be worthwhile to explore how the approaches we developed can be used for the feedback setting, and vice versa.

7.2 Long-term Future Directions

7.2.1 Richer Tasks and Evaluation

For all our experiments in this dissertation, we worked with relatively simple domains, such as a video game, table-top manipulation with a robot arm, gridworld rearrangement and 2D point navigation. While these domains allow for fast experimentation, and controlled analyses of different models, the approaches we developed in this dissertation must be built upon so they could work on more realistic tasks.

Several multimodal environments and benchmarks have been proposed recently, such as House3D [Wu et al., 2018], AI2-THOR [Kolve et al., 2017], Matterport3D [Chang et al., 2017], Habitat [Savva et al., 2019], R2R [Anderson et al., 2018b], Touchdown [Chen et al., 2019], Franka kitchen environment [Gupta et al., 2019], IKEA furniture assembly environment [Lee et al., 2021], and VirtualHome [Puig et al., 2018], which can be used to create a rich set of tasks for using language in RL and IL. While many of these environments support navigation in multiple rooms, we believe that partial observability will be an added challenge. As such, it might be worth starting with tasks that involve manipulation in a fully observable setting, or navigation with a bird's eye view of the entire arena, followed by extension to the partially observable setting.

Further, as the methods presented in this dissertation are scaled to more complex

real-world tasks, it may require learning hierarchical policies, wherein a set of low-level skills may first be learned, and the final task policy is then obtained by chaining these low-level skills. Sharma et al. [2021] propose using natural language to learn a set of skills, which can be combined with the approaches presented here.

7.2.2 Experiments on Real Robots

Once the approaches have been developed to work on more complex simulated tasks, they should be extended to work on real robots. This brings up several new challenges.

First, some of the approaches we presented require a fair amount of labeled data for the supervised learning phase. While generating data in simulation is trivial, collecting large amounts of data on real robots is time-consuming and/or expensive. As such, learning from small amounts of real-world data is imperative to scale these methods to physical robots. One approach is to start with pretrained visual and/or language encoders [He et al., 2016, Devlin et al., 2018, Radford et al., 2021]. While our experiments in this dissertation with pretrained models did not result in any performance improvement, they might be more helpful when the tasks are more realistic, since the visual and textual information in the tasks would be closer to the data distribution these models are trained on. Alternately, approaches that address the sim-to-real problem can be applied, wherein the model can first be trained in simulation, and then used on real robots with minimal finetuning [Zhao et al., 2020]. Some approaches for this include domain randomization, wherein the agent is trained on different variations of the task in simulation (e.g. colors, textures, dynamics), with the hope that it will generalize better to the unknown real-world conditions [Tobin et al., 2017]. Another line of work – grounded simulation learning – collects data in the real world to model and

correct for the errors in the simulator [Farchy et al., 2013, Hanna and Stone, 2017]. Different approaches may be more or less useful, depending on the task and the simulator.

Secondly, training a policy on real robots may result in undesirable behavior that can damage the robot or the environment, and even be dangerous for humans in the environment. As such, it is important to incorporate some safety mechanisms. Safe RL is a well-studied problem, and the approaches include optimizing for the worst-case return instead of the expected return, explicitly modelling the risk of a policy, adding constraints to the optimization process, and initializing the policy with some external knowledge or demonstrations. See Garcıa and Fernández [2015] for a more thorough review of safe RL. We already show in Chapter 6 that initializing RL with a pretrained policy is helpful. Additionally, as alluded to earlier, our approaches for imitation learning can be combined to initialize the policy for reinforcement learning. However, modelling the risk may still be required for more complex tasks, particularly if they can result in dangerous outcomes. Brown and Niekum [2019] propose a Bayesian approach that infers a distribution over reward functions, which can then be used to learn a policies with strong safety guarantees. Incorporating such an approach in the frameworks we developed may result in safe RL on real robots. The Bayesian reward inference approach also has the additional advantage that it naturally addresses the ambiguity problem in imitation learning, by reasoning about the entire distribution of possible reward functions.

7.2.3 Reward Shaping for Multi-step Tasks

In Chapters 3 and 4, we experiment with only single-step tasks, for instance, going from a start location to a goal location in Montezuma's Revenge, and changing the state

of one object in table-top robot manipulation. It would be worthwhile to explore how the approaches we developed can be extended to multi-step tasks. Importantly, this will depend on the granularity of the task descriptions provided.

If the task description involves step-by-step instruction for each subtask, then the approaches we presented can be augmented with a *Progress Predictor* module to predict if the language instruction for a subtask has been completed or not. This could then be used in conjunction with the presented approaches, where the agent starts following the first instruction, and transitions to the next one when this new module predicts that the current instruction has been completed. Reward shaping will be applied at each step using the approaches we presented in this dissertation to speed up learning. For tasks with simple subgoals, the Progress Predictor module could be heuristic-based, for instance, when the potential predicted by LEARN or PixL2R for the current state is close to 1. More complex tasks, such as those involving grasping an object as a subgoal, may need explicitly training the module to capture all the post-conditions that need to be satisfied to complete the subgoal. Semantic parsing approaches could be used to predict the formal representations of these post-conditions from natural language instructions [Kamath and Das, 2018].

On the other hand, the task description may only contain the final goal to be achieved, without the details about the subgoals. While the approaches presented in Chapters 3 and 4 can be applied directly in such cases, it is unlikely to work well, because we make a tacit assumption that the progress in the task can be inferred using relatedness of the trajectory with the provided description. This may not be true for more complex tasks, for instance, boiling water on the stove-top may first require clearing the stove-top, which may not appear related to the end goal. Huang et al. [2022] propose an approach that converts

high-level natural language descriptions into a sequence of low-level steps using pretrained large language models. This could then be combined with the approaches presented above where fine-grained instructions are available.

7.2.4 Task Adaptation for a Broader Set of Tasks

In Chapters 5 and 6, we presented approaches for language-guided task adaptation, which work on goal-based tasks. Future work could look into developing approaches that work on tasks that do not fall into this category. There are two main categories of tasks other than the types of tasks we considered in this dissertation, which could benefit from using natural language:

- Trajectory-based Tasks: These tasks require reasoning about the trajectory or the path followed by the agent. Natural language can be used to specify many of these trajectory preferences or constraints (e.g. "Stay further away from the glass tumblers", or "Keep the mug upright while moving it").

  Recently, Bucker et al. [2022] proposed a transformer-based model for task adaptation that takes in the source trajectory and language, and outputs a trajectory for the target task. While their experimental results are promising, for many tasks it may be easier to capture the intended target task using rewards as in the methods proposed in this dissertation, instead of directly modifying the trajectory. The difference between reward adaptation and trajectory adaptation may be particularly pronounced for tasks with complex transition dynamics, such as those requiring different kinds of grasps for the source and target tasks.

An important consideration for predicting the rewards for trajectory-based tasks is the parameterization of the reward function. In many trajectory-based tasks, the reward of a state depends on the relative positions of different objects, e.g., keeping a safe distance from fragile objects, or keeping a coffee mug away from electronic devices. Other factors that could affect the reward include orientation of objects (e.g. keeping a coffee mug upright) and grasping an object at different locations. A transformer-based reward model might be suitable to encode such reward functions, where the vector representations of all the entities are fed as inputs to the transformer. Different properties of the entities, such as the orientation and the current grasping location (if the object is currently grasped by the agent) can be encoded into the vector representations of the entities.

- Tasks involving State Changes: In these tasks, the agent can take actions that result in the change of state of objects in the environment, for instance, slicing an apple for a cooking task, or screwing a leg into a tabletop for a furniture assembly task. Natural language could be used to communicate modifications to these state changes (e.g. "Slice it a little more finely"), or specifying a different entity on which to apply the state change (e.g. "Screw in the black leg instead").

  For these tasks, it might be useful to explicitly learn *rules* that can be applied on entities to affect state changes. Didolkar et al. [2021] propose a neurosymbolic approach that represents entities using vectors, and rules using neural networks. Applying a rule on an entity then amounts to passing the entity vector through the network corresponding to the rule, resulting in a new vector representation (i.e. state) of the entity. They use the approach to model systems with passive dynamics, such as a falling stack of

blocks, or bouncing balls. In passive dynamics modeling, only one rule is applicable on an entity at most timesteps (e.g. for a block not in contact with any other surface, the rule corresponding to "free fall" must be applicable), and therefore, the approach proposed in Didolkar et al. [2021] trains an attention-based model to infer which rule to apply to an entity given its current vector representation. In contrast, when modeling active dynamics, multiple rules could be applicable on an entity at most timesteps, and the agent can choose to apply any of those rules. For instance, "toast" and "apply butter on" are both applicable to the entity "slice of bread", with the choice depending on the end goal. Thus, the above approach must be extended to choose a rule to apply conditioned explicitly or implicitly on the end goal. When conditioning explicitly, this would amount to first inferring the goal state for the target task, for instance, using the techniques presented in this dissertation, followed by predicting which rule to apply on each entity at different timesteps conditioned on the predicted goal state. Since linguistic descriptions for task adaptation in this setting would often involve modifications to a specific step, for instance, "add a little more salt in the third step", it may be more appropriate to directly condition the rule prediction on the source task demonstration and the linguistic description (which implicitly encapsulate the goal state for the target task).

7.2.5 Task Adaptation with Multiple Source Tasks

The task adaptation setting we considered in Chapters 5 and 6 assume a single source task. A natural extension worth looking at is the setting with multiple source tasks. This brings up several new challenges as well as possibilities, which we discuss below.

The simplest case in the setting is when the desired target task can be completely inferred from the provided language and exactly one of the source tasks. The main challenge in this case is identifying which source task to adapt from, using the provided language description. For instance, the language description may be of the form "Follow the same steps as in the task for boiling eggs, except keep the heat on for an extra minute". Hutsebaut-Buysse et al. [2019, 2020a] recently developed approaches that learn a policy for a new task given a set of base policies, where language is used to choose which base policy to use. These approaches can be used to identify the source task, and combined with approaches we presented.

A more challenging case in this setting is when the desired target task requires using information from more than one source tasks, for instance, the description may be of the form "Also use broccoli in the dish", given with respect to a demonstrated source task. However, to use broccoli the right way, the agent may need to refer to a repertoire of past tasks (which can be thought of as additional source tasks). Hammond [1986] proposed a case-based planning approach, where these past tasks are stored along with the subgoals they achieve, such as *cooking a vegetable without making it soggy*, which may then be used to identify which previous tasks are relevant for the current target task. Thus, this setting requires storing past tasks in a form they can be queried later. Further, this setting will also involve combining different aspects of source tasks to get the goal or the initial policy for the target task, which is another interesting direction to explore.

7.2.6 Language-aided Imitation Learning

As alluded to in the introduction, when humans demonstrate tasks to other humans, they often use linguistic cues to disambiguate, such as, "Turn off the heat when the water starts boiling". However, most imitation learning algorithms currently use demonstrations only, which are often incomplete and ambiguous. As such, a promising future direction is to explore using natural language with demonstrations to better infer the demonstrator's intent. We believe the approaches presented in this work would provide a useful scaffolding towards building such systems. In particular, approaches in imitation learning often involve inferring a reward function followed by reinforcement learning. By using approaches presented in Chapters 3 and 4, linguistic information could be incorporated into the inferred reward function. Further, the task adaptation setting presented in Chapters 5 and 6 requires combining information from both demonstrations and language, which is also one of the key requirements for language-aided learning from demonstrations.

An important consideration for this problem setting is that for most domains currently being used in imitation learning, the goal can often be specified using language or demonstrations alone. For instance, for the Breakout game in the Atari domain, a linguistic description of the form "Hit all the blocks with the ball" is sufficient by itself—a model that can ground this description to goal states or desirable actions doesn't need demonstrations to infer the goal state. Similarly, MuJoCo locomotion tasks [Todorov et al., 2012] can be specified using demonstrations alone, with language providing little value. Using both demonstrations and language would be important as imitation learning is scaled to more challenging tasks in the coming years, such as preparing a meal or a home repair task which involves many intricacies, some of which may be easier to communicate using demonstrations

while others may be better communicated using language.

Thus, as a first step, we need to create benchmarks that are rich enough to need both demonstrations and language for task specification, while still being solvable by current RL and IL methods in a reasonable amount of time. A good place to start may be to abstract away the low-level complexity of control and observation, by creating cooking and repairing tasks in a gridworld domain. For instance, in the cooking domain, the ingredients, utensils, and appliances may be placed in different cells of the grid, and the action space of the agent consists of picking and placing objects, slicing, stirring, turning on the microwave, etc. Many of these actions can be made parameterized, for instance, how fine to slice or how long to microwave, but could also be non-parameterized actions that need to be performed multiple times in succession to achieve the desired outcome. These tasks can be made rich enough so that neither modality is sufficient by itself. For instance, some steps like heating an ingredient until it is of a specific shade of light brown, or chopping an ingredient to get pieces with the desired size may be easier to observe from demonstrations, than communicating through language. Analogously, some higher-level goals such as moving objects to clear the kitchen counter may be better communicated using language. More generally, the demonstrations can be used to communicate *how* to perform specific steps, while language can be used to communicate *what* to do and *why*. Once we have models that work for these abstract domains, they can be scaled to more realistic tasks.

Chapter 8

Conclusion

As we usher into the age of automation, service robots that assist humans in day-to-day tasks will soon be possible. Since human environments, such as homes, office spaces, factories, and hospitals, are diverse, they could benefit from various kinds of tasks being automated with the help of these service robots. Further, different humans have their own preferences for how a task should be completed. Therefore, it is infeasible to pre-program these learning agents to know all the tasks that they might need to perform during their lifetimes, and we need techniques that allow end-users to teach new tasks to these learning agents conveniently. Reinforcement learning and imitation learning are common approaches used to specify new tasks to these agents, by providing a reward function or demonstration(s) respectively. In this dissertation, we augment these task specification modalities with natural language to make task specification easier for end users.

The following is a summary of the contributions in this dissertation:

1. **Language for Task Specification in Reinforcement Learning**

   (a) **Chapter 3**

      i. We introduce a novel setting for reinforcement learning, in which the agent is provided with a natural language description of the task, in addition to a sparse reward function.

ii. We develop a two-phase approach, which first learns a relatedness model between the agent's action distribution and natural language descriptions, using supervised learning, and then uses this relatedness model during policy learning to generate auxiliary rewards—if the agent's actions are related to the task description, the auxiliary reward is high; otherwise, the auxiliary reward is low.

iii. Extending a previous result (Ng et al. [1999]), we prove that the auxiliary rewards, when added to the rewards from the environment, do not change the optimal policy.

iv. We create a new benchmark in the Atari game Montezuma's Revenge, which consists of discrete state and action spaces. A diverse set of tasks are constructed in this domain, and our experiments on these tasks show that the proposed approach results in both a better final policy, as well as more efficient policy training.

(b) **Chapter 4**

i. We extend the approach presented in Chapter 3, wherein, the supervised learning phase is modified to use the sequence of states instead of the action distribution. This addresses several major limitations of the previous approach, such as, discarding the temporal information in sequences, discarding the information in states, and not being applicable to continuous control tasks.

ii. We introduce additional modifications to the previous approach, such as a more robust training objective, and data augmentations to handle states rep-

resented as high-dimensional images.

iii. We create a continuous control benchmark, involving a robot manipulator interacting with everyday objects, and show that the auxiliary rewards generated from natural language task descriptions by our approach result in faster policy training when the environment reward is sparse.

iv. Further, we demonstrate that our approach can be used to improve policy training efficiency even when the environment rewards are dense (but presumably suboptimal). This motivates a new paradigm for RL, wherein coarse dense rewards could be designed by hand, and then the proposed approach could be used to obtain a further boost in policy training efficiency.

2. **Language for Task Specification in Imitation Learning**

   (a) **Chapter 5**

      i. We introduce a new setting—zero-shot task adaptation using natural language. Given a demonstration of a task (the *source* task), and a natural language description between the source task and the desired task (the *target* task), the agent needs to learn the target task, without any demonstrations for the target task.

      ii. We propose an approach that decomposes the problem into two subproblems: (1) predict the goal state for the target task, using the source demonstration and the language, and (2) learn a reward or value function for the target task, given the predicted goal state.

iii. We construct a new benchmark consisting of multi-step rearrangement tasks, involving blocks in an organizer, and a diverse set of adaptations, such as inserting a step, modifying a step, and deleting a step. Our experiments show that the proposed approach successfully infers the target task in a large fraction of the test cases.

(b) **Chapter 6**

i. We extend the language-guided task adaptaion problem setting introduced in Chapter 5 to relational domains, and create two new benchmarks: (1) grid-world rearrangement, and (2) 2D navigation. Together, these domains cover various different settings in an MDP (including discrete versus continuous state and action spaces, long versus short horizon tasks, and unique optimal action sequence versus multiple optimal action sequences), and therefore serve as a good testbed for the task adaptation problem setting.

ii. We develop Relational Reward Adaptation, an approach decomposes the problem into (1) predicting the goal state for the target task, and (2) learning a distance function between two states in the environment. The goal state prediction uses a transformer-based architecture, which enables reasoning about relations between different entities present in the task. By combining these two subproblems, we can obtain a reward function for the target task, which is then used to learn a policy.

iii. We develop Relational Policy Adaptation, a new approach that assumes the agent has a policy for the source task, and learns to adapt the source policy for the target task. This is achieved by learning a transformer-based adaptation

model that takes in a state and the action under the source task, to predict the action under the target task, conditioned on the source task goal and language.

iv. We experiment with a diverse set of adaptations in the domains we introduce, most of which require reasoning about relative positions of entities. Our experiments demonstrate that: (1) Relational Reward Adaptation leads to successfully learning the target task from the source demonstration and language in many test tasks, but there is room for improvement; (2) Relational Policy Adaptation can be used to complete some target tasks without RL, but there is a significant room for improvement; and (3) combining the two approaches followed by finetuning with RL leads to a much better performance than using either approach independently.

To conclude, this dissertation introduced new problem settings, benchmark datasets, and approaches that enable incorporating natural language for task specification in sequential decision making, namely by augmenting rewards in reinforcement learning and demonstrations in imitation learning. We believe that this dissertation provides a foundation for natural language task specification in RL and IL, and will induce future research in this direction.

Bibliography

Pooya Abolghasemi, Amir Mazaheri, Mubarak Shah, and Ladislau Bölöni. Pay attention! - Robustifying a Deep Visuomotor Policy through Task-Focused Attention. *arXiv:1809.10093 [cs]*, November 2018. URL http://arxiv.org/abs/1809.10093. Reporter: arXiv:1809.10093 [cs] arXiv: 1809.10093.

Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

Dario Amodei and Jack Clark. Faulty reward functions in the wild, 2016.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086, 2018a. Reporter: Proceedings of the IEEE conference on computer vision and pattern recognition.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018b.

Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. *arXiv preprint arXiv:1711.00482*, 2017. Reporter: arXiv preprint arXiv:1711.00482.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Jacob Arkin, Matthew R Walter, Adrian Boteanu, Michael E Napoli, Harel Biggie, Hadas Kress-Gazit, and Thomas M Howard. Contextual awareness: Understanding monologic natural language instructions for autonomous robots. In *Robot and Human Interactive Communication (RO-MAN), 2017 26th IEEE International Symposium on*, pages 502–509. IEEE, 2017.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013. Reporter: Transactions of the Association for Computational Linguistics Publisher: MIT Press.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 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*, pages 751–761, 2016. Reporter: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A Knepper, and Yoav Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. *arXiv preprint arXiv:1806.00047*, 2018a. Reporter: arXiv preprint arXiv:1806.00047.

Valts Blukis, Dipendra Misra, Ross A. Knepper, and Yoav Artzi. Mapping Navigation Instructions to Continuous Control Actions with Position-Visitation Prediction. *arXiv:1811.04179 [cs]*, December 2018b. URL http://arxiv.org/abs/1811.04179. Reporter: arXiv:1811.04179 [cs] arXiv: 1811.04179.

Valts Blukis, Yannick Terme, Eyvind Niklasson, Ross A Knepper, and Yoav Artzi. Learning to Map Natural Language Instructions to Physical Quadcopter Control using Simulated Flight. *arXiv preprint arXiv:1910.09664*, 2019. Reporter: arXiv preprint arXiv:1910.09664.

Stevo Bozinovski and Ante Fulgosi. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, volume 3, pages 121–126, 1976.

SRK Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 126–135. Association for Computational Linguistics, 2012a. Reporter: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1.

SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012b. Reporter: Journal of Artificial Intelligence Research.

Kianté Brantley, Wen Sun, and Mikael Henaff. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2019.

Alexander Broad, Jacob Arkin, Nathan Ratliff, Thomas Howard, and Brenna Argall. Real-time natural language corrections for assistive robotic manipulators. *The International Journal of Robotics Research*, 36(5-7):684–698, 2017. Number: 5-7 Reporter: The International Journal of Robotics Research Publisher: SAGE Publications Sage UK: London, England.

Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

Daniel S Brown and Scott Niekum. Deep bayesian reward learning from preferences. *arXiv preprint arXiv:1912.04472*, 2019.

Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederick Jelinek, John Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *IJCAI*, pages 3352–3358, 2015.

Arthur Bucker, Luis Figueredo, Sami Haddadin, Ashish Kapoor, Shuang Ma, and Rogerio Bonatti. Reshaping robot trajectories using natural language commands: A study of multi-modal data alignment using transformers. *arXiv preprint arXiv:2203.13411*, 2022.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005(598-603):18, 1997.

Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12538–12547, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *International Conference on Acoustics, Speech, and Signal Processing,*, pages 695–698. IEEE, 1989.

John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. *arXiv preprint arXiv:1811.07882*, 2018. Reporter: arXiv preprint arXiv:1811.07882.

Michael Collins. A new statistical parser based on bigram lexical dependencies. *arXiv preprint cmp-lg/9605012*, 1996.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1070.

Yuchen Cui and Scott Niekum. Active reward learning from critiques. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6907–6914. IEEE, 2018.

Yuchen Cui, Qiping Zhang, Alessandro Allievi, Peter Stone, Scott Niekum, and W Bradley Knox. The empathic framework for task learning from implicit human feedback. *arXiv preprint arXiv:2009.13649*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Aniket Didolkar, Anirudh Goyal, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Heess, Michael C Mozer, and Yoshua Bengio. Neural production systems. *Advances in Neural Information Processing Systems*, 34, 2021.

Jin Dong, Marc-Antoine Rondeau, and William L Hamilton. Distilling structured knowledge for text-based relational reasoning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6782–6791, 2020.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017. Reporter: Advances in neural information processing systems.

Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1):7–52, 2001.

Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 39–46, 2013.

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

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. *arXiv preprint arXiv:1806.02724*, 2018.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Anirudh Goyal, Alex Lamb, Phanideep Gampa, Philippe Beaudoin, Sergey Levine, Charles Blundell, Yoshua Bengio, and Michael Mozer. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020a.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, Macao, China, August 2019.

Prasoon Goyal, Scott Niekum, and Raymond Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. In *Conference on Robot Learning*, pages 485–497. PMLR, 2020b.

Prasoon Goyal, Raymond J Mooney, and Scott Niekum. Zero-shot task adaptation using natural language. *arXiv preprint arXiv:2106.02972*, 2021.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Kristian J Hammond. Chef: A model of case-based planning. In *AAAI*, volume 86, pages 267–271, 1986.

Josiah P Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3): 335–346, 1990.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Sachithra Hemachandra, Felix Duvallet, Thomas M Howard, Nicholas Roy, Anthony Stentz, and Matthew R Walter. Learning models for following natural language directions in unknown environments. *arXiv preprint arXiv:1503.05079*, 2015.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016. Reporter: Advances in neural information processing systems.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Thomas M Howard, Istvan Chung, Oron Propp, Matthew R Walter, and Nicholas Roy. Efficient natural language interfaces for assistive robots. In *IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS) Work. on Rehabilitation and Assistive Robotics*. Citeseer, 2014a.

Thomas M Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6652–6659. IEEE, 2014b.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Fast Task-Adaptation for Tasks Labeled Using Natural Language in Reinforcement Learning. *arXiv preprint arXiv:1910.04040*, 2019. Reporter: arXiv preprint arXiv:1910.04040.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latre. Language Grounded Task-Adaptation in Reinforcement Learning. *Computational Intelligence*, page 6, 2020a. Reporter: Computational Intelligence.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Pre-trained Word Embeddings for Goal-conditional Transfer Learning in Reinforcement Learning. *arXiv:2007.05196 [cs, stat]*, July 2020b. URL http://arxiv.org/abs/2007.05196. Reporter: arXiv:2007.05196 [cs, stat] arXiv: 2007.05196.

Aishwarya Kamath and Rajarshi Das. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978*, 2018.

Isaac Kamlish, Isaac Bentata Chocron, and Nicholas McCarthy. SentiMATE: Learning to play Chess through Natural Language Processing. *arXiv preprint arXiv:1907.08321*, 2019. Reporter: arXiv preprint arXiv:1907.08321.

Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*, 2017. Reporter: arXiv preprint arXiv:1704.05539.

Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 787–798, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. *Advances in Neural Information Processing Systems*, 31, 2018.

Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *arXiv preprint arXiv:1907.03146*, 2019.

Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *The AAAI-2004 workshop on supervisory control of learning and adaptive systems*. San Jose, CA, 2004. Reporter: The AAAI-2004 workshop on supervisory control of learning and adaptive systems.

Vitaly Kurin, Sebastian Nowozin, Katja Hofmann, Lucas Beyer, and Bastian Leibe. The atari grand challenge dataset. *arXiv preprint arXiv:1705.10998*, 2017.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

Youngwoon Lee, Edward S Hu, and Joseph J Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6343–6349. IEEE, 2021.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

David M Magerman. Statistical decision-tree models for parsing. *arXiv preprint cmp-lg/9504030*, 1995.

Nikhil Mehta and Dan Goldwasser. Improving Natural Language Interaction with Robots Using Advice. *arXiv preprint arXiv:1905.04655*, 2019. Reporter: arXiv preprint arXiv:1905.04655.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. Reporter: Advances in neural information processing systems.

Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. *arXiv preprint arXiv:1809.00786*, 2018. Reporter: arXiv preprint arXiv:1809.00786.

Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015. Number: 7540 Reporter: Nature Publisher: Nature Publishing Group.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Deep transfer in reinforcement learning by language grounding. *arXiv preprint arXiv:1708.00133*, 2017. Reporter: arXiv preprint arXiv:1708.00133.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874, 2018.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

Andrew Y Ng, Stuart J Russell, and others. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000. Reporter: Icml.

Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, pages 10–15607. Berlin, Germany, 2013.

Daniel Nyga, Subhro Roy, Rohan Paul, Daehyung Park, Mihai Pomarlan, Michael Beetz, and Nicholas Roy. Grounding robot plans from natural language instructions with incomplete world knowledge. In *Conference on Robot Learning*, pages 714–723, 2018. Reporter: Conference on Robot Learning.

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-Shot Visual Imitation. *ICLR 2018*, April 2018. URL http://arxiv.org/abs/1804.08606. Reporter: ICLR 2018 arXiv: 1804.08606.

Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Foxl. Prospection: Interpretable plans from language by predicting the future. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6942–6948. IEEE, 2019. Reporter: 2019 International Conference on Robotics and Automation (ICRA).

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Deepak Ramachandran and Eyal Amir. Bayesian Inverse Reinforcement Learning. In *IJCAI*, volume 7, pages 2586–2591, 2007. Reporter: IJCAI.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *arXiv:1011.0686 [cs, stat]*, March 2011. URL http://arxiv.org/abs/1011.0686. Reporter: arXiv:1011.0686 [cs, stat] arXiv: 1011.0686.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

Akanksha Saran, Ruohan Zhang, Elaine Schaertl Short, and Scott Niekum. Efficiently guiding imitation learning algorithms with human gaze. *arXiv preprint arXiv:2002.12500*, 2020.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9339–9347, 2019.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80, 2008.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015. Reporter: International conference on machine learning.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Roger Shank and Robert Abelson. Scripts, plans, goals and understanding, 1977.

Lin Shao, Toki Migimatsu, Qiang Zhang, Kaiyuan Yang, and Jeannette Bohg. Concept2Robot: Learning Manipulation Concepts from Instructions and Human Demonstrations. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 2020. ISBN 978-0-9923747-6-1. doi: 10.15607/RSS.2020.XVI.082. URL http://www.roboticsproceedings.org/rss16/p082.pdf. Meeting Name: Robotics: Science and Systems 2020 Reporter: Robotics: Science and Systems XVI.

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. *arXiv preprint arXiv:2110.01517*, 2021.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *arXiv preprint arXiv:2010.12083*, 2020.

Theodore R. Sumers, Mark K. Ho, Robert D. Hawkins, Karthik Narasimhan, and Thomas L. Griffiths. Learning Rewards from Linguistic Feedback. *arXiv:2009.14715 [cs]*, September 2020. URL http://arxiv.org/abs/2009.14715. Reporter: arXiv:2009.14715 [cs] arXiv: 2009.14715.

Jaeyong Sung, Seok Hyun Jin, and Ashutosh Saxena. Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *Robotics Research*, pages 701–720. Springer, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Pradyumna Tambwekar, Andrew Silva, Nakul Gopalan, and Matthew Gombolay. Interpretable Policy Specification and Synthesis through Natural Language and RL. *arXiv:2101.07140 [cs]*, January 2021. URL http://arxiv.org/abs/2101.07140. Reporter: arXiv:2101.07140 [cs] arXiv: 2101.07140.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, volume 1, page 2, 2011.

Jesse Thomason, Jivko Sinapov, Maxwell Svetlik, Peter Stone, and Raymond J. Mooney. Learning Multi-Modal Grounded Linguistic Semantics by Playing" I Spy". In *IJCAI*, pages 3477–3483, 2016. Reporter: IJCAI.

Jesse Thomason, Aishwarya Padmakumar, Jivko Sinapov, Justin Hart, Peter Stone, and Raymond J Mooney. Opportunistic active learning for grounding natural language descriptions. In *Conference on Robot Learning*, pages 67–76. PMLR, 2017.

Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence-video to text. In *Proceedings of the IEEE international conference on computer vision*, pages 4534–4542, 2015.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

H. J. Austin Wang and Karthik Narasimhan. Grounding Language to Entities and Dynamics for Generalization in Reinforcement Learning. *arXiv:2101.07393 [cs]*, January 2021. URL http://arxiv.org/abs/2101.07393. Reporter: arXiv:2101.07393 [cs] arXiv: 2101.07393.

Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6629–6638, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology, 1971.

Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018.

Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4651–4659, 2016.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

Allan Zhou, Vikash Kumar, Chelsea Finn, and Aravind Rajeswaran. Policy architectures for compositional generalization in control. *arXiv preprint arXiv:2203.05960*, 2022.

Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.