SCALABLE AND INTERPRETABLE APPROACHES FOR LEARNING TO FOLLOW NATURAL LANGUAGE INSTRUCTIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Dipendra Kumar Misra

May 2019

SCALABLE AND INTERPRETABLE APPROACHES FOR LEARNING TO FOLLOW NATURAL LANGUAGE INSTRUCTIONS

Dipendra Kumar Misra, Ph.D.

Cornell University 2019

Agents that can execute natural language instructions have many applications. For example, an assistive house robot that can follow instructions will reduce the time spent on doing household chores. Natural language provides a convenient medium for users to express a wide variety of objectives for these agents. However, to achieve this goal the agent must understand the meaning of natural language instruction, reason about its context, and take appropriate actions. In this thesis, we will introduce new instruction following tasks along with new approaches. The presented approach focuses on designing scalable and interpretable agents that can follow complex natural language instructions. We also introduce an integrated learning framework for instruction following that contains an implementation of several tasks and approaches.

BIOGRAPHICAL SKETCH

Dipendra Misra received his bachelors from the Indian Institute of Technology, Kanpur (IITK) in 2013 majoring in computer science. He was an OPJEMS scholar for the year 2011-2012 and 2012-2013. He finished his undergrad thesis on the topic of *designing agents that can solve IQ problems* under Amitabha Mukerjee (IITK) and Sumit Gulwani (Microsoft Research, Redmond). During his undergraduate studies he also did research on space-bounded complexity classes, a distributed learning framework for collaborative drawing, and designing efficient CUDA implementations for path planning and object detection for use in the autonomous vehicle project at Carnegie Mellon University.

He joined the PhD program at Cornell University in August 2013 receiving the university fellowship. He was a visiting researcher at the Stanford AI Laboratory at Stanford University from Fall 2014 to Summer of 2015. His PhD research was supervised by Yoav Artzi. During his PhD he interned with the natural language processing group at Microsoft Research, Redmond (Summer 2017) and with the reinforcement learning group at Microsoft Research, New York (Fall 2018). During his PhD, he worked on grounded language understanding, semantic parsing, model-based reinforcement learning and designing theoretical provable reinforcement learning algorithms. He is active in the program committees of natural language understanding and machine learning. At the 2018 conference of Association of Computation Linguistic (ACL 2018), he co-organized the third edition of the workshop on Representation Learning for Natural Language Understanding (Rep4NLP 2018).

Dedicated to

Frederic Chopin

Polish Composer and Virtuoso Pianist (1810-1849)

ACKNOWLEDGEMENTS

My research in the past few years has been made possible due to the support and help from various people who in their own way made my doctoral journey both productive and memorable. Firstly, I am thankful to my advisor Yoav Artzi for his guidance over the past few years. Yoav provided me the research freedom to pursue challenging problems in the field of natural language understanding and his advise was critical to my progress.

I am also thankful to my collaborators and friends at Microsoft Research including John Langford, Akshay Krishnamurthy, Mikael Bruce Henaff, Adith Swaminathan, Alekh Agarwal, Miro Dudik, Bill Dolan, and Chris Quirk for valuable research discussion. I was fortunate to have done two internship at Microsoft Research.

I am indebted to Percy Liang (Stanford), Scott Yih (Allen Institute for Artificial Intelligence), Ming-Wei Chang (Google Research) and Xiaodong He (JD AI Research) for their mentorship and research collaboration. Their experience was of great help and I learned a lot from them about research in natural language understanding.

I owe a lot to Kavosh Asadi and Michael Littman from Brown University. I met Kavosh during my internship at Microsoft Research and he became an important research collaborator. Together we pursued and published several ideas in model-based reinforcement learning.

I am thankful to my peers at Cornell University including Andrew Bennett, Ryan Benmalek, Akshay Bhat, Ashudeep Singh, Valts Blukis, Tianze Shi, Arzoo Katiyar, Alane Suhr, Max Grusky, Ashesh Jain, Ozan Sener, Hema Koppula and the Cornell NLP research group for useful discussion and providing feedback on my research work. I am also thankful for the opportunity to mentor amaz-

ing students: Kevin Lee, Kejia Tao, Jiaqi Su, Shivam Bharuka, Michela Meister, Claudia Yan, Marwa Mouallem, Eyvind Niklasson, and Max Shatkhin.

My family has been a constant source of strength and I would not have been here without their support, encouragement and the values they imparted me. I am grateful to my partner Nga Than for her care and counsel which made difficult times seem easier. I am fortunate to have Shubham Toshniwal, Yin Huang, Siddharth Gaur, Howard Chen, Longqi Yang, Yin Cui, Yuhang Zhao, and Neta Tamir as my friends. It is quite easy for me to loose track of time in their company. Finally, I am grateful to the citizenry of New York city, a city dearest to me, for giving me some of the best time of my life.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . iv
Acknowledgements . . . v
Table of Contents . . . vii
List of Tables . . . x
List of Figures . . . xi

1 Introduction to the task of Instruction Following 1
1.1 Introduction . . . 1
1.2 Problem Definition . . . 1
1.3 Motivating Applications . . . 3
1.4 Challenges . . . 5
1.5 Overview . . . 7

2 Background 8
2.1 Models . . . 8
2.1.1 Conditional Random Field . . . 8
2.1.2 Neural Network Models . . . 10
2.2 Learning Algorithms . . . 13
2.2.1 Supervised Learning (Behavior Cloning) . . . 14
2.2.2 Imitation Learning . . . 15
2.2.3 Multi-Armed Bandit and Contextual Bandit . . . 16
2.2.4 Reinforcement Learning . . . 18

3 Related Work 22
3.1 Rule-based Approach . . . 22
3.2 Grammar Based Approaches . . . 22
3.3 Graphical Model Approaches . . . 24
3.4 Neural Network Approaches . . . 25
3.5 Related Tasks . . . 26

4 Learning to Follow High Level Instructions 29
4.1 Introduction . . . 29
4.2 Problem Statement . . . 32
4.3 Approach Overview . . . 33
4.3.1 Representation . . . 33
4.3.2 Formal Overview . . . 35
4.4 Anchored Verb Lexicons . . . 36
4.5 Semantic Parsing Model . . . 39
4.6 Lexicon Induction from Training Data . . . 41
4.7 Environment-Driven Lexicon Induction at Test Time . . . 42
4.8 Inference and Parameter Estimation . . . 44

4.9 Dataset and Experiments . . . 46
4.9.1 Dataset . . . 46
4.9.2 Experiments and Results . . . 46
4.10 Conclusion . . . 49

5 A Single Model Approach **51**
5.1 Introduction . . . 51
5.2 Technical Overview . . . 54
5.3 Model . . . 56
5.4 Learning . . . 58
5.5 Reward Shaping . . . 63
5.6 Experimental Setup . . . 66
5.7 Results . . . 69
5.8 Conclusions . . . 72

6 An Interpretable Model for Instruction Following **74**
6.1 Introduction . . . 74
6.2 Technical Overview . . . 77
6.3 Model . . . 78
6.4 Learning . . . 83
6.5 Tasks and Data . . . 85
6.5.1 LANI . . . 85
6.5.2 CHAI . . . 87
6.6 Experimental Setup . . . 89
6.7 Results . . . 91
6.8 Discussion . . . 95

7 Cornell Instruction Following Framework (CIFF) **97**
7.1 Introduction . . . 97
7.2 Features of CIFF . . . 98

8 Conclusion **100**
8.1 Future Directions . . . 101

A Appendix for Chapter 4 **105**
A.1 Parsing Text into Control Flow Graph. . . . 105
A.2 Dataset: Samples and Challenges . . . 106
A.3 Examples of Planning and Simulation . . . 109
A.4 Mapping Object Descriptions . . . 110
A.5 Manual Rules for Parsing Conditions . . . 111
A.6 Feature Equation . . . 112
A.7 Assignment Problem . . . 115

B Appendix for Chapter 5 **118**
B.1 Reward Shaping Theorems 118
B.2 Evaluation Systems 119
B.3 Parameters and Initialization 121
B.3.1 Architecture Parameters 121
B.3.2 Initialization 122
B.3.3 Learning Parameters 122
B.4 Dataset Comparisons 123
B.5 Common Questions 124

C Appendix for Chapter 6 **128**
C.1 Tasks and Data: Comparisons 128
C.2 Reward Function 128
C.3 Baseline Details 130
C.4 Hyperparameters 131
C.5 CHAI Error Analysis 131
C.6 Examples of Generated Goal Prediction 132

LIST OF TABLES

4.1 Some lexical entries for the verb *turn* 38
4.2 New verbs and concepts induced at test time (Section 4.7). Posconditions denote the learned logical form representing the meaning of the underlined verb. 45
4.3 Results on the metrics and baselines described in section 4.9.2. The numbers are normalized to 100 with larger values being better. 48

5.1 Corpus statistics for the block environment we use and the SAIL navigation domain. 68
5.2 Mean and median (Med.) development results. 69
5.3 Mean and median (Med.) test results. 69

6.1 Summary statistics of the two corpora. 85
6.2 Qualitative analysis of the LANI and CHAI corpora. We sample 200 single development instructions from each corpora. For each category, we count how many examples of the 200 contained it and show an example. 88
6.3 Performance on the development data. 91
6.4 Performance on the held-out test dataset. 92
6.5 Development goal prediction performance. We measure distance (Dist) and accuracy (Acc). 92
6.6 Mean goal prediction error for LANI instructions with and without the analysis categories we used in Table 6.2. The *p*-values are from two-sided *t*-tests comparing the means in each row. 93

B.1 Comparison of several related natural language instructions corpora. Size denotes the number of instructions in the dataset. AIL is the Average Instruction Length and ATL is the Average Trajectory Length. 123

C.1 Comparison of LANI and CHAI to several existing natural language instructions corpora. Size denotes the number of instructions in the dataset. AIL is the Average Instruction Length and ATL is the Average Trajectory Length. 128
C.2 Mean goal prediction error for CHAI instructions with and without the analysis categories we used in Table 6.2. The *p*-values are from two-sided *t*-tests comparing the means in each row. 132

LIST OF FIGURES

1.1 An example of the instruction following task. The PR2 robot is given an instruction by the user to make the Affogato dessert. . . 2

2.1 A conditional random field with 6 random variables: A, B, C, D, E, F. The CRF models the probability of assigning values to the random variable set $Y = \{B, C, E, F\}$ given $X = \{A, D\}$. 9
2.2 A sample neural network with 2 hidden layers. The neural network takes an input $x \in \mathbb{R}^{128}$ and outputs a probability distribution over the two values in $\{0, 1\}$. 11

4.1 A lexicon learned on the training data cannot possibly cover all the verb-concept mappings needed at test time. Our algorithm learns the meaning of new verbs (e.g., *fill*) using the environment context. 30
4.2 Graphical model overview: we first deterministically shallow parse the text $\bar{x}$ into a control flow graph consisting of shallow structures $\{c_i\}$. Given an initial environment s_1, our semantic parsing model maps these frame nodes to logical forms $\{z_i\}$ representing the postconditions. From this, a planner and simulator generate the action sequences $\{a_i\}$ and resulting environments $\{s_i\}$. 30
4.3 We deterministically parse text into a shallow structure called a control flow graph. 35
4.4 Logical forms for given clauses c_{i-1} and c_i, environment s_i, and previous logical form z_{i-1} are generated from both a lexicon induced from training data and a test-time search procedure based on the environment. 44

5.1 Instructions in the Blocks environment. The instructions all describe the same task. Given the observed RGB image of the start state (large image), our goal is to execute such instructions. In this task, the direct-line path to the target position is blocked, and the agent must plan and move the Toyota block around. The small image marks the target and an example path, which includes 34 steps. 52
5.2 Illustration of the policy architecture showing the 10th step in the execution of the instruction *Place the Toyota east of SRI* in the state from Figure 5.1. The network takes as input the instruction $\bar{x}$, image of the current state I_{10}, images of previous states I_8 and I_9 (with $K = 2$), and the previous action a_9. The text and images are embedded with LSTM and CNN. The actions are selected with the task specific multi-layer perceptron. 56

5.3 Visualization of the shaping potentials for two tasks. We show demonstrations (blue arrows), but omit instructions. To visualize the potentials intensity, we assume only the target block can be moved, while rewards and potentials are computed for any block movement. We illustrate the sparse problem reward (left column) as a potential function and consider only its positive component, which is focused on the goal. The middle column adds the distance-based potential. The right adds both potentials. 64
5.4 Mean distance error as a function of the ratio of training examples that include complete trajectories. The rest of the data includes the goal state only. 71

6.1 Example instructions from our two tasks: LANI (left) and CHAI (right). LANI is a landmark navigation task, and CHAI is a corpus of instructions in the CHALET environment. 75
6.2 An illustration for our architecture (Section 6.3) for the instruction *turn left and go to the red oil drum* with a LINGUNETdepth of $m = 4$. The instruction $\bar{x}$ is mapped to $\bar{\mathbf{x}}$ with an RNN, and the initial panorama observation I_P to $\mathbf{F}_0$ with a CNN. LINGUNETgenerates $\mathbf{H}_1$, a visual representation of the goal. First, a sequence of convolutions maps the image features $\mathbf{F}_0$ to feature maps $\mathbf{F}_1,\ldots,\mathbf{F}_4$. The text representation $\bar{\mathbf{x}}$ is used to generate the kernels $\mathbf{K}_1,\ldots,\mathbf{K}_4$, which are convolved to generate the text-conditioned feature maps $\mathbf{G}_1,\ldots,\mathbf{G}_4$. These feature maps are de-convolved to $\mathbf{H}_1,\ldots,\mathbf{H}_4$. The goal probability distribution P_g is computed from $\mathbf{H}_1$. The goal location is the inferred from the max of P_g. Given l_g and p_t, the pose at step t, the goal mask $\mathbf{M}_t$ is computed and passed into an RNN that outputs the action to execute. 79
6.3 Segmented instructions in the LANI domain. The original reference path is marked in red (start) and blue (end). The agent, using a drone icon, is placed at the beginning of the path. The follower path is coded in colors to align to the segmented instruction paragraph. 87
6.4 Scenario and segmented instruction from the CHAI corpus. . . . 88
6.5 Likert rating histogram for expert human follower and our approach for LANI. 93
6.6 Goal prediction probability maps P_g overlaid on the corresponding observed panoramas I_P. The top example shows a result on LANI, the bottom on CHAI. 95

A.1 Sample of 3D Environments that we consider. Environments consists of several objects, each object can have several states. Different environment have different set of objects with different configuration. There can be more than one objects of the same category. 107

C.1 Goal prediction probability maps P_g overlaid on the corresponding observed panoramas I_P. The top three examples show results from LANI, the bottom three from CHAI. The white arrow indicates the forward direction that the agent is facing. The success/failure in the LANI examples indicate if the task was completed accurately or not following the task completion (TC) metric. 133

CHAPTER 1

INTRODUCTION TO THE TASK OF INSTRUCTION FOLLOWING

1.1 Introduction

Natural language is a convenient medium for humans to communicate a broad range of objectives to Artificial Intelligence (AI) agents. As a result, the problem of instruction following has been studied from the early days of AI as a scientific discipline [Winograd, 1972]. Figure 1.1 shows an example of the instruction following problem. In this example, the PR2 robot is given an instruction by the user to make a dessert: *"Take some coffee in a cup. Add ice cream of your choice. Finally add raspberry syrup to the mixture"*. In order to successfully complete the task, the robot must understand the instruction, reason about its environment, and generate a sequence of actions to execute the task. In this thesis, we will introduce new instruction following tasks and datasets, and propose new models, learning algorithms and a software framework for the instruction following problem.

1.2 Problem Definition

Let $\mathcal{S}$ be the set of all possible world states, i.e., all the different worlds in which the agent could be in. At a given time, the agent is in some world state $s \in \mathcal{S}$. Let $\mathcal{X}$ be the set of all possible instructions. A natural language instruction $\bar{x} \in \mathcal{X}$ is as a sequence of tokens $\bar{x} = (x_1, \cdots, x_m)$ where for every $i \in \{1, 2, \cdots, m\}$, $x_i \in \mathcal{V}$ for some vocabulary $\mathcal{V}$. The agent takes action from a set $\mathcal{A}$. $\mathcal{A}$ can be discrete

User Instruction: *"Take some coffee in a cup. Add ice cream of your choice. Finally add raspberry syrup to the mixture."*

Figure 1.1: An example of the instruction following task. The PR2 robot is given an instruction by the user to make the Affogato dessert.

or continuous control or a mixture of both. We will assume there is a special action $\texttt{STOP} \in \mathcal{A}$ which indicates task completion. There is a transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, such that on taking an action $a \in \mathcal{A}$ in world state $s \in \mathcal{S}$, the new world state changes to $s' \sim T(. \mid s, a)$. In this thesis, we will assume a deterministic transition function. For a deterministic transition function taking an action a in the world state s always leads to the world state $T(s, a) \in \mathcal{S}$. We further assume without loss of generality that $T(s, \texttt{STOP}) = s$ for every $s \in \mathcal{S}$.

Given an instruction $\bar{x}$, the agent takes a sequence of actions to generate an execution $\bar{e} = \{(s_1, a_1), (s_2, a_2), \cdots, (s_n, a_n)\}$ where $s_{i+1} = T(s_i, a_i)$ for all $i \in \{1, 2, \cdots, n-1\}$ and $a_n = \texttt{STOP}$. In practice, the agent does not always have access to the world state. For example, the world state contains information about the configuration of all objects in the world, whereas the agent may only have access to images of the world. We therefore distinguish between a world state s and an *agent context* $\tilde{s}$. An agent context $\tilde{s}$ is the information the agent has access

to at each step in order to make decisions. We will denote $\tilde{\mathcal{S}}$ to be the set of all possible agent contexts. At a given time step t, the agent selects the next action a_t using the current agent context $\tilde{s}_t$.

Let $\mathcal{E}$ be the set of all possible executions. Then there exists an underlying utility function $\mathcal{U} : \mathcal{E} \times \mathcal{X} \to \mathbb{R}$, that given an instruction $\bar{x}$ and an execution $\bar{e}$, assigns a score $\mathcal{U}(\bar{x}, \bar{e})$. This score is a measure of how well the execution $\bar{e}$ followed the given natural language instruction $\bar{x}$. The exact form of the utility function can vary between applications. For example, for applications where the only concern is to reach a destination, the utility function is only dependent upon the final state in the execution. The aim of an instruction following agent is to generate an execution $\bar{e}$ given an instruction $\bar{x}$, that maximizes the value of $\mathcal{U}(\bar{e}, \bar{x})$.

1.3 Motivating Applications

There are many applications of an agent that can follow natural language instructions. We will consider four specific applications: an agent assisting humans with household chores, a drone aiding in delivering supplies, a natural language interface for autonomous vehicles, and a natural language interface for computer applications. The first three applications focus on robotic agents, whereas the last application is an example of a non-robotic agent.

House Assistive Robots Assistive robots that can do house chores such as laundry, cleaning dishes or retrieving objects are of great value. This would be particularly useful to those who need assistance with daily routine tasks.

While robots have been used for specific household chores such as folding towels [Maitin-Shepard et al., 2010] and unloading items from a dishwasher [Saxena et al., 2008], in general, people would expect their robot to do a variety of tasks to justify the price. Natural language allows user to express broad open-ended objectives to their assistive robots and to enable them to do a variety of tasks.

Delivery Drone The ability of drones to navigate in open land space and either pick or deliver packages have made them suitable for various applications. Unsurprisingly several initiatives have been undertaken by the industry to use drones for various commercial purposes [dro]. In general, a user commanding a drone may not have the Global Positioning System (GPS) coordinates of the destination but they may have a description of the goal. This motivates the use of natural language for giving commands to the drone. For example, a user may specify: *"give this medical supply to a wounded person who is somewhere north of the lake"*. Even when the GPS information is available, it might be difficult to precisely locate the user due to inherent inaccuracy with GPS. Natural language could be used to bridge the gap by giving instructions to improve the accuracy. For example, a user may say, *"I am located across the street from where you are"*.

Autonomous Vehicles (AVs) Autonomous vehicles have been successfully demonstrated in real world environment [Urmson et al., 2008]. However, currently most of these vehicles can only accept a destination on the map. In general, people may have a more complex requirement or can change their mind in the middle of a trip. For example, a user may remember that they have to shop for a certain item or need a coffee. Natural language could be used to communicate these new objectives to the car. For example, a person may say, *"I just*

remembered to pick my child from the school. Can you stop at St. Xavier on the way?". Natural language can make this communication seamless and enjoyable for the user.

Interface for Computer Applications It can often take humans a significant amount of time to complete a task when using a computer application. Even when the user knows how to accomplish the task, it can still take significant time and effort in writing the relevant program. For example, consider a spreadsheet application where the user wishes to add a new column for every spreadsheet in a specific folder and populate the new column with the initials of the customers. It takes non-trivial amount of time to write a script to do this task. However, it can often be convenient to express this objective using natural language. This motivates the need for natural language driven interfaces for computer applications. For example, the user can express the above task using a simple two sentence description, *"add a new column for every spreadsheet in the folder data. Populate this column with the initials of the value in the customer column."* Such interfaces can greatly boost the workplace productivity and allow users to focus on tasks requiring more creativity.

1.4 Challenges

An agent following natural language instructions has to solve challenges in natural language understanding, context reasoning (such as reasoning about the environment) and planning.

Language Understanding Challenges Natural language understanding poses several challenges for an instruction following agent including: understanding spatial relations (e.g., *"pick the book* ***to the left of the*** *coffee mug"*), high level instructions (e.g., *"****prepare*** *me a coffee"*), resolving co-references (e.g., *"take the mug and put* ***it*** *in the sink"*) and understanding comparatives (e.g., *"pick the* ***smaller*** *of the two can and dispose it in the trash"*).

Context Understanding Challenges The context for an instruction following agent varies based on the task. For example, the context for a robot following instructions is the physical environment where the robot is situated. The agent may be able to perceive the context only in the form of high dimensional observation like an RGB image. In this case, the agent must understand the objects present in the environment, be able to infer their attributes (such as their color and size), and be able to ground natural language entities such as *brown book* or *white mug* to these objects. This context reasoning becomes more challenging when the world is partially observable, i.e., the agent can only perceive a small part of the environment at a given time.

Planning Challenges The agent has to search in an exponentially large search space, of size $|\mathcal{A}|^H$, in order to generate the right sequence of actions. Even for a simple application the action space could be of size 4 and the horizon H of length 20 giving rise to a search space of size 10^{12}. Further, for many applications it maybe impractical to use efficient dynamic programming method or even heuristic search methods. For example, while beam search is widely used for natural language understanding problems [Andor et al., 2016], a robot cannot practically do beam search in real life.

Learning Challenges An agent trained on a set of instructions must not only generalize to new instructions but also new world states during testing. Further, it is challenging to scale instruction following datasets since collecting instructions require users to interact with the agent. This limits the ability to scale to millions of examples, like in text generation, dialogues and machine translation. Thus, we need learning algorithms that can train agents to generalize to new instructions and new world states using few training examples.

1.5 Overview

The rest of this thesis is organized as follows. In Chapter 2 we cover the fundamental technical details needed for understanding the results in the thesis. Chapter 3 covers previous work on the instruction following problem. In Chapter 4 we present our approach for following high-level instructions in a simulated house environment where we assume access to underlying world state. In Chapter 5 we present a single-model method for instruction following that directly works with raw sensory data and a sample-efficient learning algorithm. In Chapter 6 we present an interpretable model for navigation and manipulation in 3D environments. In Chapter 7 we present the Cornell Instruction Following Framework (CIFF) which contains several tasks, models, and learning algorithms for instruction following. Finally, in Chapter 8 we conclude by listing open questions and directions for future work.

CHAPTER 2

BACKGROUND

In this chapter we provide the necessary technical background for understanding the results in this thesis. Most existing approaches for the instruction following task consists of a model class (also known as the hypothesis class) and a learning algorithm that interacts with the world to find an accurate model in the class. In general, the model class could use different forms of input and the learning algorithm could utilize various forms of feedback. Below we review common model classes and learning algorithms used in the instruction following literature.

2.1 Models

Two common models that occur in the instruction following literature are conditional random fields [Misra et al., 2015] and neural network models [Mei et al., 2016b, Misra et al., 2017]. We briefly review them below.

2.1.1 Conditional Random Field

A Conditional Random Field (CRF) is used to express a probability distribution $P(Y \mid X)$ where X, Y are a set of random variables. Since we are modeling a conditional distribution, we always know the value of random variables in X and our aim is to infer the value of random variables in Y. The CRF encodes the independency assumptions between the variables in $X \cup Y$ in the form of an undirected graph. The nodes of this graph are the random variables in $X \cup Y$.

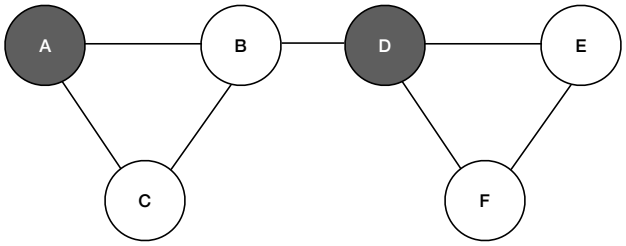


Figure 2.1: A conditional random field with 6 random variables: A, B, C, D, E, F. The CRF models the probability of assigning values to the random variable set $Y = \{B, C, E, F\}$ given $X = \{A, D\}$.

For every random variable $v \in Y$ we have $P(v|Y - \{v\}, X) = P(v|\partial v, X)$ where ∂v is the set of all neighbors of v. For example, Figure 2.1 shows a CRF graph. In this example, the random variable B is independent of F and E given values of random variables A, C and D.

Let C be the set of maximal cliques of the CRF. In Figure 2.1 the set $\{A, B, C\}$ and $\{D, E, F\}$ constitute the two set of maximal cliques. Each maximal clique $c \in C$ in the CRF has an associated function called the factor function ϕ_c. Let $Z_c \subseteq X \cup Y$ be the set of random variables that are member of c. The factor function ϕ_c takes in the random variables in Z_c as input and outputs a scalar value $\phi_c(Z_c)$. Using the factor notation, we can express $P(Y \mid X) \propto \exp\{\sum_{c \in C} \phi_c(Z_c)\}$. The constant factor in this proportionality is called the partition function and is given by $\exp\{\sum_{c \in C} \phi_c(Z_c(Y', X))\}$ where $Z_c(Y', X)$ denotes the random variable in clique c with values assigned in Y', X.

Inference The inference in CRF can be challenging in general however exact inference is often possible for simple graphs like trees and chain. Loop belief propagation is a general technique for an arbitrary CRF. We refer interested authors to Section 8.4 in Bishop [2006] for details.

Learning The factor functions can be parameterized giving rise to a set of parameterized probability distribution $P_\theta(Y \mid X)$ where θ is the set of parameters of all factor functions. Given access to a dataset $\{(X_i, Y_i)\}_{i=1}^N$, we can estimate the parameters using maximum likelihood. Formally, we optimize the following objective:

$$\max_\theta \sum_{i=1}^{N} \ln P_\theta(Y_i \mid X_i).$$

A difficult procedure in this optimization is approximating the partition function. This is often performed with the aid of some heuristic approximation like beam search. The beam search procedure returns a set of value assignments for Y. The partition function is then approximated by summing over these assignments.

2.1.2 Neural Network Models

Neural networks represent a category of hypothesis class that allows performing rich non-linear transformation on the input. A simple example of a neural network model for doing binary classification is shown in Figure 2.2. Given an input $x \in \mathbb{R}^{128}$, the model performs a sequence of two non-linear transformation. Each non-linear transformation consists of an affine transformation followed by applying the element-wise ReLu non-linearity. The element-wise ReLu non-linearity transforms the input vector $v \in \mathbb{R}^d$ to $\text{ReLu}(v) \in \mathbb{R}^d$ where $\text{ReLu}(v)_i = \max\{0, v_i\}$. The output of a non-linear transformation is called a hidden representation and the dimensionality of the output is called its width (or number of neurons). In our example, the network applies the first non-linear transformation on x to generate the hidden representation $h_1 \in \mathbb{R}^{64}$. It then

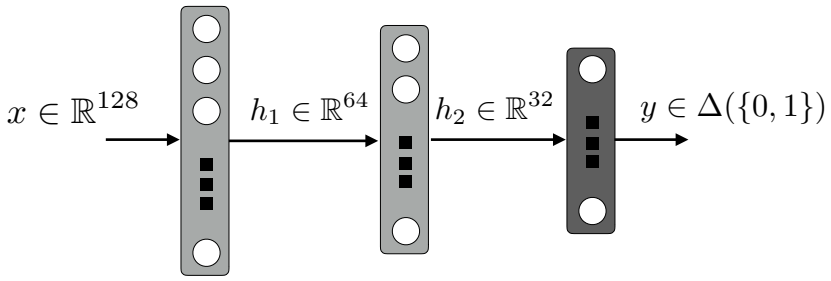


Figure 2.2: A sample neural network with 2 hidden layers. The neural network takes an input $x \in \mathbb{R}^{128}$ and outputs a probability distribution over the two values in $\{0, 1\}$.

applies the second non-linear transformation on h_1 to generate the hidden representation $h_2 \in \mathbb{R}^{32}$. Finally, it applies an affine transformation on h_2 to generate $z \in \mathbb{R}^2$ followed by a softmax operation on z to generate a probability distribution over 0 and 1. Softmax maps an input $z \in \mathbb{R}^d$ to a discrete probability distribution: $\left(\frac{\exp(z_1)}{\sum_{j=1}^{d} \exp(z_j)}, \cdots, \frac{\exp(z_d)}{\sum_{j=1}^{d} \exp(z_j)}\right)$. In general, models could use different number and type of non-linear transformations.

In the architecture we saw above, we apply a sequence of many non-linear transformations on the input. Such models are called *deep neural networks*. It has been empirically observed that *deep neural networks* are able to provide the right structural bias for several problems in artificial intelligence. This finding has been surprising since theoretically a two layer neural network can represent any continuous function to arbitrary precision provided it has enough neurons. However, two-layer networks haven't performed well on many important problems. We refer interested readers to Goldberg [2016] for a more thorough exposition.

Inference A neural network can be viewed as a directed acyclic graph where each node of the graph performs a unit of computation on the input from the

incoming edges, and outputs the result along the outgoing edges. An efficient inference procedure then follows from doing a topological sort on this graph and feeding the input and the result of previous computation to the next node in the sorted list. The final result is the output of the last node in the sorted list.

Learning Neural networks are generally trained using first order methods which only require the gradient of the loss with respect to the parameters. These approaches include stochastic gradient descent, Adagrad learning [Duchi et al., 2010], RMSPROP [Tieleman and Hinton, 2012], and Adam optimization [Kingma and Ba, 2014]. The gradient is computed using an efficient procedure called backpropagation [Rumelhart et al., 1988]. The backpropagation procedure requires doing inference on the neural network given an input. The output of the network is used to compute a scalar loss value. This loss value is used to compute the gradient with respect to the output of the final computation unit. The gradients are then propagated to the previous units using the chain rule from calculus. In our example from Figure 2.2, let $y^\star$ be the gold output corresponding to an input x. This gives us a cross entropy loss of $\ell = -\ln P(y^\star \mid x)$. We can then use the chain rule to compute the gradient of the loss with respect to the output of all computation units giving us:

$$\begin{aligned}
\nabla_z \ell &= \nabla_{P(.|x)} \ell \; J(P(. \mid x), z) \\
\nabla_{h_2} \ell &= \nabla_z \ell \; J(z, h_2), \\
\nabla_{h_1} \ell &= \nabla_{h_2} \ell \; J(h_2, h_1),
\end{aligned}$$

where $J(f, g)$ is the Jacobian matrix of f with respect to g and $J(f, g)_{ij} = \frac{\partial f_i}{\partial g_j}$. The gradient of the loss with respect to the parameters θ of a computation unit can be computed using $\nabla_\theta \ell = \nabla_h \ell \odot J(h, \theta)$, where $\nabla_h \ell$ is the gradient of the loss

with respect to the output of the computation unit h. The Jacobian matrix $J(h, \theta)$ can be locally computed using the knowledge of the computation performed by the unit.

2.2 Learning Algorithms

Instruction following can be viewed as a sequential decision making problem. An instruction following example can be described using a Markov Decision Process $(\mathcal{S}, \mathcal{A}, T, R, H, \mu)$ where $\mathcal{S}$ is a set of world state, $\mathcal{A}$ is a set of actions, $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a transition function which generates probability distribution over the next state given the current state and action, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function that takes the current state and action and returns a scalar reward, H is the maximum number of actions the agent is allowed to take in any single episode and $\mu \in \Delta(\mathcal{S})$ is a distribution over the starting state. [1] Agent does not observe the world state but instead receives an agent context $\tilde{s}$ from a high-dimensional agent context space $\tilde{\mathcal{S}}$. We will assume that there exists an encoding function $f : \mathcal{S} \to \Delta(\tilde{\mathcal{S}})$ such that given a state $s \in \mathcal{S}$ the agent observes a context $\tilde{s} \sim f(. \mid s)$.

A policy $\pi : \tilde{\mathcal{S}} \to \Delta(\mathcal{A})$ takes an agent context as input and generates probability distribution over actions. The aim of an learning algorithm is to find a policy from a set of policies Π that maximizes the expected total reward J^π where:

$$J^\pi = E_{s_1 \sim \mu(.), \tilde{s}_t \sim f(.|s_t), a_t \sim \pi(.|\tilde{s}_t), s_{t+1} \sim T(.|s_t, a_t)} \left[\sum_{t \geq 1} R(s_t, a_t) \mid \pi, \mu \right]. \tag{2.1}$$

A policy π induces a distribution $d^\pi(.)$ over the agent context. Formally,

[1] For a given set $\mathcal{U}$, $\Delta(\mathcal{U})$ represents the set of all possible probability distribution over $\mathcal{U}$

$d^{\pi}(\tilde{s}) = \frac{1}{H}\sum_{h=1}^{H} P_h(\tilde{s} \mid \pi, \mu)$ where $P_h(\tilde{s} \mid \pi, \mu)$ is the probability of observing $\tilde{s}$ after h actions when the start state is sampled from μ and actions are sampled from π. It is straightforward to verify that d^{π} is a distribution and its value is higher for contexts that are more likely to be visited by following π.

2.2.1 Supervised Learning (Behavior Cloning)

Supervised learning algorithm (also known as behaviour cloning) assumes access to an oracle policy $\pi^{\star}$ and optimizes the following objective:

$$\max_{\theta} \mathbb{E}_{\tilde{s}\sim d^{\star}(.), a\sim\pi^{\star}(.|\tilde{s})} [\ln \pi_{\theta}(a \mid \tilde{s})] \tag{2.2}$$

where π_{θ} is the policy being trained with parameters θ and $d^{\star}$ is the distribution induced over contexts by $\pi^{\star}$. In practice, this objective is optimized using a set of demonstration $\mathcal{D} = \{(\tilde{s}_1^{(i)}, a_1^{(i)}, \tilde{s}_2^{(i)}, a_2^{(i)}, \cdots, \tilde{s}_{m_i}^{(i)}, a_{m_i}^{(i)})\}_{i=1}^{N}$ which gives us:

$$\max_{\theta} \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m_i} \ln \pi_{\theta}(a_j^{(i)} \mid \tilde{s}_j^{(i)})$$

The objective could be regularized with L2 penalty of the weight giving us:

$$\max_{\theta} \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m_i} \ln \pi_{\theta}(a_j^{(i)} \mid \tilde{s}_j^{(i)}) - \lambda|\theta|_2,$$

or using the entropy of the policy giving us:

$$\max_{\theta} \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m_i} \left\{ \ln \pi_{\theta}(a_j^{(i)} \mid \tilde{s}_j^{(i)}) - \lambda \sum_{a\in\mathcal{A}} \pi_{\theta}(a \mid \tilde{s}_j^{(i)}) \ln \pi_{\theta}(a \mid \tilde{s}_j^{(i)}) \right\}.$$

In both cases $\lambda \geq 0$ is a hyperparameter controlling the regularization.

Supervised learning however suffers from *exposure bias* phenomenon which arises from mismatch between the train time state distribution (as generated by demonstrations) and the test time state distribution (as generated by agent's policy). During training the agent is being trained to optimize equation 2.2. However, during testing the agent is being evaluated on the following objective:

$$\mathbb{E}_{\tilde{s}\sim d^{\pi}(.),a\sim\pi^{\star}(.|\tilde{s})}\left[\ln \pi_{\theta}(a \mid \tilde{s})\right].$$

In general, $d^{\pi}(.)$ and $d^{\star}(.)$ can be very different and therefore an agent trained on contexts sampled from $d^{\star}(.)$ needn't do very well on contexts sampled from $d^{\pi}(.)$. To counter this problem several alternative exists ranging from imitation learning and contextual bandits to more general reinforcement learning based solutions. We briefly review them next.

2.2.2 Imitation Learning

When we have access to the oracle policy $\pi^{\star}$ during training then we can use more efficient imitation learning based algorithms. The aim of the imitation learning algorithms is to mimic the behaviour of the oracle policy. Unlike supervised learning where we only have access to a set of demonstrations generated by the oracle policy, the imitation learning algorithms assume that the oracle policy can be accessed at any point of the training. While imitation learning algorithms perform better than supervised learning they require a stricter assumption which is not easy to satisfy in practice. Dataset Aggregation (DAGGER) is an example of an imitation learning algorithm that is widely used in practice [Ross et al., 2011].

DAGGER performs a sequence of T rounds. At the beginning of round t,

DAGGER creates a mixture policy $\pi_t = \beta_t \pi^\star + (1-\beta_t)\hat{\pi}_{t-1}$ by combining the agent's current policy $\hat{\pi_{t-1}}$ and $\pi^\star$ using mixing coefficient β_t. The policy $\hat{\pi}_0$ is chosen as a randomly initialized policy. The mixture policy π_t is used to sample actions to explore the context space. DAGGER maintains a dataset D of the context $\tilde{s}$ visited by the agent and the oracle action $\pi^\star(\tilde{s})$ for this context. The dataset D is maintained across all rounds (hence the name Dataset Aggregation). At the end of round t, we compute the new agent policy $\hat{\pi}_t$ by solving the following optimization:

$$\hat{\pi}_t = \arg\max_{\pi \in \Pi} \sum_{(\tilde{s},a) \in D} \ln \pi(a \mid \tilde{s}).$$

The value of mixing coefficients β_t is generally chosen as p^{t-1} for some value $p \in (0, 1)$. This ensures that the agent samples action using the oracle policy at the beginning of training (similar to supervised learning) but gradually relies more on the agent's policy for sampling actions. This helps in minimizing the exposure bias problem. For more details on imitation learning algorithms we refer interested readers to Daumé et al. [2009], Ross and Bagnell [2014] and Ross et al. [2011].

2.2.3 Multi-Armed Bandit and Contextual Bandit

A multi-armed bandit problem consists of a set of K actions (or arms) and K distributions $D_i \in \Delta([0, 1]), \forall i \in \{1, 2, \cdots, K\}$. The task takes place in rounds where in each round the agent takes an action and receives a reward. The task automatically completes after T rounds. If in round t the agent takes the action a_t then it receives a reward $r \sim D_{a_t}(.)$. Let μ_i be the mean reward for the distribution D_i. The agent takes action using a policy $\pi \in \Pi$ that can depend upon the actions

taken in previous rounds and the reward received for taking those actions. The aim is to design a policy π that minimizes the expected regret $Reg_T(\pi)$ after T rounds:

$$Reg_T(\pi) = \max_{i \in K} \mu_i T - E\left[\sum_{t=}^{T} r_t(\pi)\right], \tag{2.3}$$

where $r_t(\pi)$ is the reward received at time t when actions are taken according to π. The multi-armed bandit problem has been studied thoroughly in the literature and algorithms exist for designing efficient policies [Auer et al., 2002a]. However, multi-armed bandit setting is quite restrictive for real life problems. An extension to the multi-armed bandit was therefore proposed by Langford and Zhang [2008] called the contextual bandit problem. Formally, a contextual bandit setting can be defined using the tuple $(\tilde{\mathcal{S}}, D, \mathcal{A}, R)$ where $\tilde{\mathcal{S}}$ is a set of all agent contexts, $D \in \Delta(\tilde{\mathcal{S}})$ is a distribution over the agent contexts, $\mathcal{A}$ is a set of actions and $R : \tilde{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Given a context $\tilde{s} \in \tilde{\mathcal{S}}$, the agent takes an action $a \in \mathcal{A}$ and receives a reward $R(\tilde{s}, a)$. The problem is to find a policy $\pi : \tilde{\mathcal{S}} \to \Delta(\mathcal{A})$ that maximizes the expected reward J^{π}_{cb} received by the agent where:

$$J^{\pi}_{cb} = E_{\tilde{s} \sim D(.), a \sim \pi(.|\tilde{s})}[R(\tilde{s}, a)].$$

Several algorithms have been proposed for designing efficient policies [Agarwal et al., 2014]. We will briefly describe a simple algorithm proposed in Krishnamurthy et al. [2016a]. Let $D = \{(\tilde{s}_i, a_i, p_i, r_i)\}_{i=1}^{n}$ be a training dataset of size n where $\tilde{s}_i$ is the agent context for the i^{th} example, a_i is the action taken by the agent with probability p_i and $r_i = R(\tilde{s}_i, a_i)$. The algorithm then learns a model $f_\theta : \tilde{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ where $f_\theta(\tilde{s}, a)$ is an estimator for $R(\tilde{s}, a)$. The model is learned by solving the following regression problem:

$$\min_{\theta} \sum_{i=1}^{n} p_i \left(f_\theta(\tilde{s}_i, a_i) - r_i\right)^2$$

For more details on contextual bandit literature we refer interested authors to Agarwal et al. [2014] and Krishnamurthy et al. [2016a].

2.2.4 Reinforcement Learning

Reinforcement learning is a class of algorithms where the agent learns to take actions by interacting with the environment in order to maximize the expected total reward objective. This allows the agent to recover from exposure bias problem with supervised learning while being more general than the contextual bandit setting and less restrictive than imitation learning setting. In reinforcement learning, the objective is to learn a policy π to maximize the expected total reward objective J^π (Equation 2.1).

In order to analyze reinforcement learning algorithms it is convenient to define two value functions Q^π and V^π for a policy π. Formally, $V^\pi : \mathcal{S} \to \mathbb{R}$ where:

$$V^\pi(s) = \mathbb{E}_{\tilde{s}_t \sim f(.|s_t), a_t \sim \pi(.|\tilde{s}_t), s_{t+1}=T(s_t,a_t)} \left[\sum_{t=1}^{H} R(s_t, a_t) \mid s_1 = s \right]$$

and $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ where:

$$Q^\pi(s, a) = \mathbb{E}_{\tilde{s}_t \sim f(.|s_t), a_t \sim \pi(.|\tilde{s}_t), s_{t+1}=T(s_t,a_t)} \left[\sum_{t=1}^{H} R(s_t, a_t) \mid s_1 = s, a_1 = a \right]$$

It is easy to verify that $J^\pi = \mathbb{E}_{s \sim \mu(.)}[V^\pi(s)]$ and $V^\pi(s) = \mathbb{E}_{\tilde{s} \sim f(.|s), a \sim \pi(.|\tilde{s})}[Q^\pi(s, a)]$. A fundamental result in reinforcement learning theory states that for every Markov Decision Process there exists a deterministic optimal policy $\pi^\star$ such that

for every $s \in \mathcal{S}$ and $\pi \in \Pi$ we have $V^{\pi^\star}(s) \geq V^{\pi}(s)$. This directly implies $J^{\pi^\star} \geq J^{\pi}$ for every $\pi \in \Pi$. We will define $V^\star = V^{\pi^\star}$ and $Q^\star = Q^{\pi^\star}$. It can be easily verified that $\pi^\star(\tilde{s}) = \arg\max_{a \in \mathcal{A}} Q^\star(s, a)$ where $\tilde{s} \sim f(. \mid s)$.

Reinforcement learning algorithms are either model-based: they learn a model of the reward function and transition dynamics and use it to take actions, or they are model-free: they directly learn the optimal policy without learning the model. We will focus on model-free algorithms and refer interested authors to Asadi et al. [2018] for discussion on model-based methods. Broadly there are two class of model-free reinforcement learning algorithms: value function based algorithms and policy gradient algorithms.[2] The value function based algorithms learn the optimal Q-function $Q^\star$ and use it to take actions by exploiting the relation $\pi^\star(\tilde{s}) = \arg\max_{a \in \mathcal{A}} Q^\star(s, a)$. In order to learn $Q^\star$, value function based approaches rely on the Bellman optimality condition which states:

$$Q^\star(s, a) = R(s, a) + \max_{a' \in \mathcal{A}} Q^\star(T(s, a), a').$$

Let $Q_\theta : \tilde{\mathcal{S}} \times \mathcal{A} \to \mathbb{R}$ be a model with parameters θ. The model is trained to minimize the following squared loss error:

$$\min_\theta \mathbb{E}_{\tilde{s}, a, R, \tilde{s}'} \left[\left(Q_\theta(\tilde{s}, a) - R - \max_{a' \in \mathcal{A}} Q_\theta(\tilde{s}', a') \right)^2 \right], \tag{2.4}$$

where R is the reward received for taking action a in the agent context $\tilde{s}$ and $\tilde{s}'$ is the agent context after taking action a. Samples for minimizing the above objective are generally collected using a policy that balances between exploration and exploitation such as the ϵ-greedy policy or the Boltzmann policy. For example, ϵ-greed policy will with probability ϵ take an action randomly uniformly from $\mathcal{A}$ and with probability $1 - \epsilon$ will take the action greedily according

[2]This is a simplification and in general there are algorithms that use ideas from both value function methods and policy gradient methods.

to the current model Q_θ. The Boltzmann policy samples action using the distribution $p_{boltzmann}(a \mid s) \propto \exp\{\frac{Q_\theta(s,a)}{\tau}\}$ where τ is a hyperparameter controlling the exploration and exploitation tradeoff.

It can be shown that $Q^\star$ is the only solution to the Bellman optimality condition. Therefore, if we can solve the optimization in Equation 2.4 then we are guaranteed that the learned value function is $Q^\star$. This approach is called Q-learning with function approximation or deep Q-learning when Q_θ is a deep neural network [Mnih et al., 2013].

In contrast to value based methods, policy gradient methods directly parameterize the policy and train it to optimize the objective J^π. Let π_θ be a policy with parameters θ then policy gradient methods use gradient descent optimization for learning θ giving rise to the following update equations:

$$\theta \leftarrow \theta - \eta \nabla_\theta J^{\pi_\theta}_{RL}.$$

where η is the learning rate. The gradient term $\nabla_\theta J^{\pi_\theta}_{RL}$ is computed using the policy gradient theorem [Sutton et al., 1999] which gives us:

$$\nabla_\theta J^{\pi_\theta} = H\mathbb{E}_{\tilde{s}\sim d^\pi(.), a\sim\pi(.|\tilde{s})}[\nabla \ln \pi(a \mid \tilde{s}) Q^\pi(a, \tilde{s})].$$

The above gradient is approximated using an experience which is a sequence of context, action and rewards: $\{(\tilde{s}_1, a_1, R_1), (\tilde{s}_2, a_2, R_2), \cdots, (\tilde{s}_H, a_H, R_H)\}$ where $a_i \sim \pi(. \mid \tilde{s}_i)$ and taking action a_i in $\tilde{s}_i$ gives a reward of R_i and a new context of $\tilde{s}_{i+1}$. We approximate the gradient using an experience:

$$\nabla_\theta J^{\pi_\theta} \approx \sum_{t=1}^{H} \nabla \ln \pi(a_t \mid \tilde{s}_t) Q^\pi(a_t, \tilde{s}_t)].$$

The $Q^{\pi}(a_t, \tilde{s}_t)$ is either approximated using the sum of rewards $\sum_{t' \geq t} R_{t'}$ giving rise to the REINFORCE algorithm or approximated using another model giving rise to the actor critic algorithm. For more details on policy gradient methods, we refer the readers to Sutton and Barto [2018].

CHAPTER 3

RELATED WORK

3.1 Rule-based Approach

Rule based approaches for instruction following rely on a set of rules for natural language understanding and planning and assume access to underline world state. Early work on instruction follow used rule-based systems. For example, SHRLDU [Winograd, 1972] is a rule-based system for following instructions to move blocks in a 3D world. However, rule-based approaches can only handle examples covered by the rules. Further, scaling these rules to complex tasks is extremely challenging. This realization gave rise to data-driven approaches for instruction following which can be roughly classified into: grammar based approaches, graphical model approaches, and neural network based approaches.

3.2 Grammar Based Approaches

Grammar based approaches use a formal grammar and a lexicon to map instructions to formal representations. This is also known as the problem of *semantic parsing*. For example, consider the following lambda calculus formal representation: $\lambda e.\texttt{move-to}(e)$. This representation denotes a function that takes an argument e and applies the predicate `move-to` to this argument. In order to use this formal representation for instruction following it has to be executable, i.e., the predicate on applying the argument should produce a result. For example, applying the predicate `move-to` to an argument e representing an object in the

environment could result in the agent moving to that object. The set of predicates constitute an *ontology* which is either provided as input or learned.

Grammar-based approaches have been widely studied for natural language understanding tasks including instruction following. For example Chen and Mooney [2011] and Kim and Mooney [2012] learnt a semantic parser using context-free grammars, mapping instructions from the SAIL corpus [MacMahon et al., 2006] to navigation plans. The SAIL corpus includes instruction for navigating in a 3D environment. In the SAIL task the world contains several objects such as a chair and a lamp, and the agent's perception of the world is a symbolic representation of its first-person visual field of view.

Grammar formalisms other than context-free grammars have also been applied in the literature. For example, Matuszek et al. [2012b] use Combinatory Categorial Grammars (CCG) [Steedman and Baldridge, 2003] for mapping instructions to typed lambda calculus expressions representing the generated navigation plan. They train a semantic parser using a set of instructions with gold-standard lambda calculus annotations. On the other hand Artzi and Zettlemoyer [2013] and Artzi et al. [2014] instead proposed mapping navigation instructions in the SAIL corpus to navigation plans using weak supervision. Rather than assuming access to gold-standard lambda calculus annotations, they use task completion accuracy.

In contrast to the above methods relying on a compositional grammars like CFG or CCG, MacGlashan et al. [2015] used the IBM models from the statistical machine translation literature for mapping instructions to reward functions. A planning algorithm then generates an action sequence maximizing the reward. Similarly, Andreas and Klein [2015] used alignment models to generate action

sequence from instructions for the SAIL task.

In general, while grammar based approaches can handle complex instructions, they require the use of an executable ontology which requires expensive engineering effort to design.

3.3 Graphical Model Approaches

Graphical model approaches use conditional random fields to model the relationship between the instruction, the environment, and the action sequence. Tellex and Roy [2009] introduced Generalized Grounding Graphs (G^3) to infer actions for a forklift robot. The G^3 approach parses the instruction into atomic clauses called *spatial description clauses* (SDC). Each SDC contains a figure, relation, and a variable set of landmarks that are extracted from the instruction. The G^3 models the relation between the SDCs and the grounding using a conditional random field. A grounding can be an action sequence, an object or a location in the world. They assume access to a semantic map to define the groundings, which contains symbolic information about the environment. Doing inference in G^3 requires searching over all possible robot motions which can be challenging. Howard et al. [2014] proposed an improvement to the G^3 approach called the Distributed Correspondence Graph (DCG) approach. This approach infers constraints and preferences for robot motion plans using a conditional random field as opposed to directly inferring the plan. A trajectory planner is then used to infer the action sequence from these constraints. The runtime of the DCG approach was further improved by the Hierarchical Distributed Correspondence Graph (HDCG) approach [Chung et al., 2015] which restricts the search space

for the DCG.

While graphical model approaches have been applied to robot instruction following [Tellex and Roy, 2009, Misra et al., 2014], and learning semantic maps from natural language descriptions [Walter et al., 2014, Duvallet et al., 2016], they require various assumptions such as access to semantic maps, access to a syntactic parser, and domain-specific feature engineering. For more complex problems, it can be challenging to satisfy these assumptions.

Branavan et al. [2009, 2010] use log-linear models to map instructions to actions. This model denotes a policy for mapping agent context to an action. The parameters of this policy are trained using a policy gradient-based reinforcement learning algorithm. Similarly, Vogel and Jurafsky [2010] use log-linear models to represent the value functions associated with mapping instructions to actions. They use SARSA, a reinforcement learning algorithm based on dynamic programming, to learn the optimal value functions. These approaches require significant effort in feature engineering and have only been evaluated on simple domains.

3.4 Neural Network Approaches

Neural network methods map instructions to vector representations, which are used for generating actions. In contrast to grammar-based approaches and rule-based methods, neural network methods generally require less engineering effort. For example, no ontology or feature engineering is required.

Mei et al. [2016a] use a neural network to map instructions and environment representations to action sequences for the SAIL task. However, they use a bag-of-words based symbolic representation of the environment which is hard to scale to more complex domains. In contrast, visual observations of the environment are both easily realizable in practice and provide richer information about the world. Nguyen et al. [2018], Xiong et al. [2018], Anderson et al. [2018] and Fried et al. [2018] focus on mapping instructions and visual observations to navigation actions. Our approach in Chapter 5 and 6 uses neural networks for mapping instructions and visual observations to actions for navigation and manipulation.

Instruction following tasks have also been studied with synthetic language. Synthetic instructions are not written by humans but generated using a grammar or rule-based approach. Chaplot et al. [2018], Hermann et al. [2017] and Oh et al. [2017] introduced different neural network methods for mapping synthetic instructions and visual observations to discrete actions. Similarly Blukis et al. [2018a] provide a method for mapping synthetic instructions and visual observations to continuous control for a simulated AirSim drone [Shah et al., 2018]. However, these approaches greatly simplify the language understanding challenges and provide only an upper bound on performance with natural language instructions.

3.5 Related Tasks

Several tasks related to instruction following have also been studied in the literature. For example Matuszek et al. [2012a] learnt a joint model for mapping

instructions describing a set of objects in an image to the selected objects. This is also known as the problem of *object retrieval*. Their approach uses the CCG formalism for mapping instructions to lambda calculus expressions. The predicates in their lambda calculus expressions represent classifiers for attributes like color and size. The semantic parser and the parameters of the classifiers are jointly trained using the expectation maximization algorithm [Dempster et al., 1977]. Their approach relies on a seed training set with full supervision for initializing the model. On the other hand Krishnamurthy and Kollar [2013] proposed a more expressive formalism for object retrieval that does not require any seed training set. In addition Guadarrama et al. [2014] focused on the problem of open-vocabulary descriptions for object retrieval. Their approach uses existing datasets and resources for handling a broad range of descriptions, including ImageNet [Deng et al., 2009], IQ Engine [IQE], Google Image Search [Goo], and FreeBase [Bollacker et al., 2008]. Kazemzadeh et al. [2014] proposed a crowdsourcing approach for collecting datasets containing natural language descriptions of objects in images. They introduce a two-player game for collecting and verifying referring expressions. In contrast to using RGB images, Kong et al. [2014] proposed a graphical model solution for object retrieval in RGB-D images. Kitaev and Klein [2017] proposed a neural network-based approach for locating regions in 3D space given natural language instructions. However, they evaluate their model on simulated environments.

The opposite problem, that is generating descriptions for a given action sequence, has also been studied in the literature. This problem finds its use in generating an utterance asking for clarification or help. Tellex et al. [2014] proposed a graphical model approach for generating text asking for help. Given a desired action, their approach generates a response that is likely to make the

G^3 instruction following model take that action. Daniele et al. [2017] learn to generate instructions for guiding humans in unknown environments to follow a path, given the path specification. Mao et al. [2016] proposed a method that can generate natural language descriptions of specific objects or regions in an image, and also map descriptions to objects in the image.

Instruction following tasks often require performing complex planning. Designing agents that can do complex planning has been widely studied in the literature. Recently, deep reinforcement learning-based approaches have been proposed for playing games [Mnih et al., 2013, Guo et al., 2014, Mnih et al., 2016, Hessel et al., 2017], solving memory puzzels [Oh et al., 2016], and target-driven navigation [Zhu et al., 2017]. In Chapter 5 we present a learning algorithm inspired by work in deep reinforcement learning.

Finally, recent work has also considered the related task of *embodied question answering*. In this task, the agent is asked a question such as *"is there milk in the fridge?"* and the agent has to take a sequence of actions to gather information to answer the question. Das et al. [2018] and Gordon et al. [2018] focus on answering questions in simulated 3D house environments. These approaches use a hierarchical model which can perform both navigation and question answering. However, they only evaluate on synthetic instructions.

CHAPTER 4

LEARNING TO FOLLOW HIGH LEVEL INSTRUCTIONS

4.1 Introduction

Instructions that contain high level concepts can be challenging for an instruction following agent. For example, instructions containing verbs such as *microwave* denoting high-level concepts, which correspond to more than 10 low-level symbolic actions such as `grasp`. In this setting, it is common to find new verbs requiring new concepts at test time. For example, in Figure 4.1, suppose that we have never seen the verb *fill*. Can we impute the correct interpretation, and moreover seize the opportunity to learn what *fill* means in a way that generalizes to future instructions?

Previous work in semantic parsing handles lexical coverage in one of two ways. Kwiatkowski et al. [2010] induces a highly constrained CCG lexicon capable of mapping words to complex logical forms, but it would have to skip new words (which in Figure 4.1 would lead to microwaving an empty cup). Others [Berant and Liang, 2014] take a freer approach by performing a search over logical forms, which can handle new words, but the logical forms there are much simpler than the ones we consider.

In this paper, we present an hybrid approach that uses a lexicon to represent complex concepts but also strongly leverages the environment to guide the search space. The environment can provide helpful cues in several ways:

- Only a few environments are likely for a given scenario—e.g., the text is

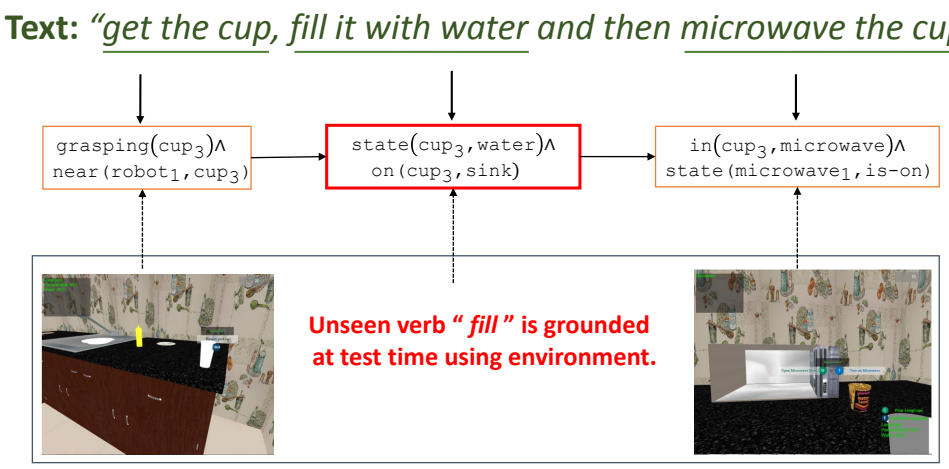


Figure 4.1: A lexicon learned on the training data cannot possibly cover all the verb-concept mappings needed at test time. Our algorithm learns the meaning of new verbs (e.g., *fill*) using the environment context.

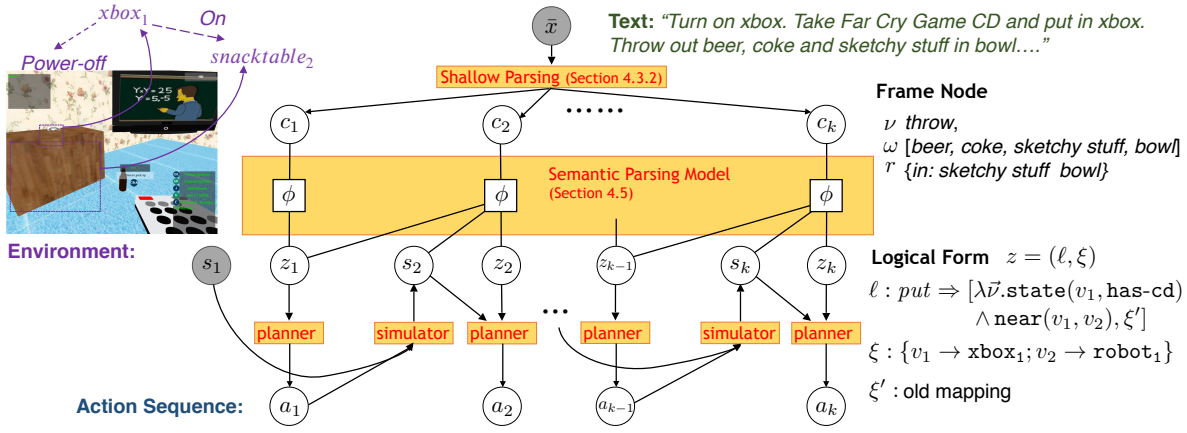


Figure 4.2: Graphical model overview: we first deterministically shallow parse the text $\bar{x}$ into a control flow graph consisting of shallow structures $\{c_i\}$. Given an initial environment s_1, our semantic parsing model maps these frame nodes to logical forms $\{z_i\}$ representing the postconditions. From this, a planner and simulator generate the action sequences $\{a_i\}$ and resulting environments $\{s_i\}$.

unlikely to ask the robot to microwave an empty cup or put books on the floor.

- The logical form of one segment of text constrains that of the next segment—e.g., the text is unlikely to ask the robot to pick a cup and then put it back immediately in the same spot.

We show that this environment context provides a signal for inducing new lexical entries that map previously unseen verbs to novel concepts. In the example in Figure 4.1, the algorithm learns that microwaving an empty cup is unlikely and this suggests that the verb *fill* must map to actions that end up making the cup not empty.

Another contribution of this paper is using postconditions as logical forms rather than actions, as in previous work [Artzi and Zettlemoyer, 2013, Misra et al., 2014]. Postconditions not only reduce the search space of logical forms, but are also a more natural representation of verbs. We define a conditional random field (CRF) model over postconditions, and use a planner to convert postconditions into action sequences and a simulator to generate new environments.

At test time, we use the lexicon induced from the training data, but also perform an environment-guided search over logical forms to induce new lexical entries on-the-fly. If the predicted action sequence uses a new lexical entry generated by the search, it is added to the lexicon, where it can be reused in subsequent test examples.

We evaluate our algorithm on a new corpus containing text commands for a household robot. The two key findings of our experiments are: First, the en-

vironment and task context contain enough information to allow us to learn lexical entries for new verbs such as *distribute* and *mix* with complex semantics. Second, using both lexical entries generated by a test-time search and those from the lexicon induced by the training data outperforms the two individual approaches. This suggests that environment context can help alleviate the problem of having a limited lexicon for grounded language acquisition.

4.2 Problem Statement

At training time, we are given a set of examples $D = \{(\bar{x}^{(m)}, s^{(m)}, a^{(m)}, \beta^{(m)})\}_{m=1}^{M}$, where $\bar{x}^{(m)}$ is a text containing natural language instructions, $s^{(m)}$ is an initial environment, $a^{(m)}$ is a human-annotated sequence of actions, and $\beta^{(m)}$ specifies a monotonic alignment between segments of $\bar{x}^{(m)}$ and segments of $a^{(m)}$. For example, given words $\bar{x}^{(m)} = x_1 x_2$ and $a^{(m)} = a_1 a_2 a_3$, $\beta^{(m)}$ might specify that x_1 aligns to $a_1 a_2$ and x_2 aligns to a_3.

At test time, given a sequence of text-environment pairs as input $\{(\bar{x}^{(n)}, s^{(n)})\}_{n=1}^{N}$, we wish to generate a sequence of actions $a^{(n)}$ for each input pair. Note that our system is allowed to use information about one test example to improve performance on subsequent ones. We evaluate a system on its ability to recover a human-annotated sequence of actions.

4.3 Approach Overview

Figure 4.2 shows our approach for mapping text $\bar{x}$ to actions $a_{1:k}$ given the initial environment s_1.

4.3.1 Representation

We use the following representation for the different variables in Figure 4.2.

Environment. An environment s_i is represented by a graph whose nodes are objects and edges represent spatial relations between these objects. We consider five basic spatial relations: `near`, `grasping`, `on`, `in` and `below`. Each object has an instance ID (e.g., $\texttt{book}_9$), a category name (e.g., *chair, xbox*), a set of properties such as *graspable, pourable* used for planning and a set of boolean states such as `has-water`, `at-channel3`, whose values can be changed by robot actions. The robot is also an object in the environment. For example, the objects $\texttt{xbox}_1$, $\texttt{snacktable}_2$, are two objects in s_1 in Figure 4.2 with relation `on` between them.

Postconditions. A postcondition is a conjunction of atoms or their negations. Each atom consists of either a spatial relation between two objects (e.g., $\texttt{on}(\texttt{book}_9, \texttt{shelf}_3)$) or a state and a value (e.g., $\texttt{state}(\texttt{cup}_4, \texttt{has-water})$). Given an environment s, the postcondition evaluates to true or false.

Actions. Each action in an action sequence a_i consists of an action name with a list of arguments (e.g., $\texttt{grasp}(\texttt{xbox}_1)$). The action name is one of 15 values ($\texttt{grasp}, \texttt{moveto}, \texttt{wait}$, etc.), and each argument is either an object in the environ-

ment (e.g., $\texttt{xbox}_1$), a spatial relation (e.g., $\texttt{in}$ for $\texttt{keep}(\texttt{ramen}_2, \texttt{in}, \texttt{kettle}_1)$, or a postcondition (e.g., for $\texttt{wait}(\texttt{state}(\texttt{kettle}_1, \texttt{boiling}))$).

Logical Forms. The logical form z_i is a pair (ℓ, ξ) containing a lexical entry ℓ and a mapping ξ. The lexical entry ℓ contains a parameterized postcondition such as $\lambda\vec{\texttt{v}}.\texttt{grasping}(\texttt{v}_1, \texttt{v}_2) \wedge \neg\texttt{near}(\texttt{v}_3, \texttt{v}_2)$, and ξ maps the variables $\vec{\texttt{v}}$ to objects in the environment. Applying the parameterized postcondition on ξ yields a postcondition; note that a postcondition can be represented by different logical forms. A lexical entry contains other information which are used for defining features, which is detailed in Section 4.4.

Control Flow Graphs. Following previous work [Tellex et al., 2011, Misra et al., 2014], we convert the text $\bar{x}$ to a shallow representation. The particular representation we choose is a *control flow graph*, which encodes the sequential relation between atomic segments in the text. Figure 4.3 shows the control flow graph for an example text. In a *control flow graph*, each node is either a frame node or a conditional node. A frame node represents a single clause (e.g., "*change the channel to a movie*") and has at most one successor node. Specifically, a frame node consists of a verb v (e.g., *arrange*, *collect*), a set of object descriptions $\{\omega_i\}$ which are the arguments of the verb (e.g., *the guinness book*, *movie channel*), and spatial relations r between the arguments (e.g., *between*, *near*). The object description ω is either an anaphoric reference (such as "*it*") or a tuple containing the main noun, associated modifiers, and relative clauses.

A conditional node contains a logical postcondition with at most one existentially quantified variable (in contrast to a frame node, which contains natural language). For example, in Figure 4.3 the conditional node contains the expression corresponding to the text *"if any of the pots has food"* There are two

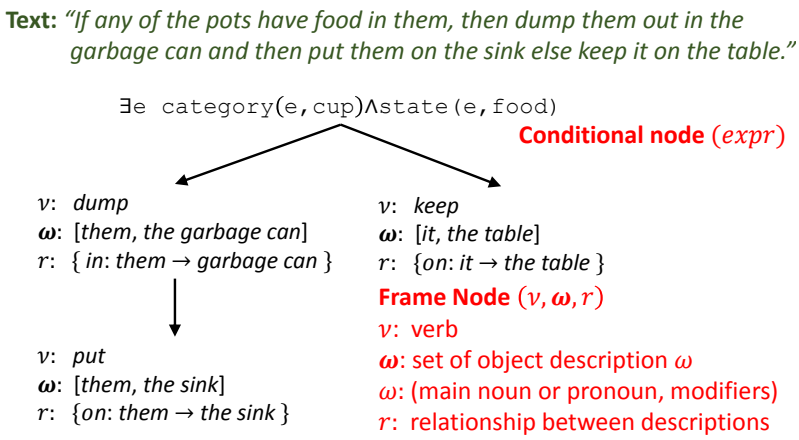


Figure 4.3: We deterministically parse text into a shallow structure called a control flow graph.

types of conditional nodes: branching and temporal. A branching conditional node represents an "*if*" statement and has two successor nodes corresponding to whether the condition evaluates to true or false in the current environment. A temporal conditional node represents an "*until*" statement and waits until the condition is false in the environment.

4.3.2 Formal Overview

Shallow Parsing. We deterministically convert the text $\bar{x}$ into its *control flow graph* G using a set of manual rules applied on its constituency parse tree from the Stanford parser [Klein and Manning, 2003]. Conditionals in our dataset are simple and can be converted into postconditions directly using a few rules, unlike the action verbs (e.g., "*fill*"), which is the focus of this paper. The details of our shallow parsing procedure is described in the appendix.

Given an environment s_1, G is reduced to a single sequence of frame nodes $c_1, \ldots, c_k$, by evaluating all the branch conditionals on s_1.

Semantic Parsing Model. For each frame node c_i and given the current environment s_i, the semantic parsing model (Section 4.5) places a distribution over logical forms z_i. This logical form z_i represents a postcondition on the environment after executing the instructions in c_i.

Planner and Simulator. Since our semantic representations involve postconditions but our model is based on the environment, we need to connect the two. We use planner and a simulator that together specify a deterministic mapping from the current environment s_i and a logical form z_i to a new environment s_{i+1}. Specifically, the planner takes the current environment s_i and a logical form z_i and computes the action sequence $a_i = \mathit{planner}(s_i, z_i)$ for achieving the post condition represented by z_i.[1] The simulator takes the current environment s_i and an action sequence a_i and returns a new environment $s_{i+1} = \mathit{simulator}(s_i, a_i)$.

4.4 Anchored Verb Lexicons

Like many semantic parsers, we use a lexicon to map words to logical forms. Since the environment plays an central role in our approach, we propose an *anchored verb lexicon*, in which we store additional information about the environment in which lexical entries were previously used. We focus only on verbs since they have the most complex semantics; object references such as "*cup*" can be mapped easily, as described in Section 4.5.

More formally, an anchored verb lexicon Λ contains lexical entries ℓ of the

[1] We use the symbolic planner of Rintanen [2012] which can perform complex planning. For example, to pick up a bottle that is blocked by a stack of books, the planner will first remove the books before grasping the bottle. In contrast, Artzi and Zettlemoyer [2013] use a simple search over implicit actions.

following form: $[v \Rightarrow (\lambda\vec{v}.S, \xi)]$ where, v is a verb, S is a postcondition with free variables $\vec{v}$, and ξ is a mapping of these variables to objects. An example lexical entry is: $[pour \Rightarrow (\lambda v_1 v_2 v_3.S, \xi)]$, where:

$S = \texttt{grasping}(\texttt{v}_1, \texttt{v}_2) \wedge \texttt{near}(\texttt{v}_1, \texttt{v}_3) \wedge \neg\texttt{state}(\texttt{v}_2, \texttt{milk}) \wedge \texttt{state}(\texttt{v}_3, \texttt{milk})$

$\xi = \{\texttt{v}_1 \rightarrow \texttt{robot}_1, \texttt{v}_2 \rightarrow \texttt{cup}_1, \texttt{v}_3 \rightarrow \texttt{bowl}_3\}$ (anchoring)

As Table 4.1 shows, a single verb will in general have multiple entries due to a combination of polysemy and the fact that language is higher-level than postconditions.

Advantages of Postconditions. In contrast to previous work [Artzi and Zettlemoyer, 2013, Misra et al., 2014], we use postconditions instead of action sequence for two main reasons. First, postconditions generalize better. To illustrate this, consider the action sequence for the simple task of filling a cup with water. At the time of learning the lexicon, the action sequence might correspond to using a tap for filling the cup while at test time, the environment may not have a tap but instead have a pot with water. Thus, if the lexicon maps to action sequence, then it will not be applicable at test time whereas the postcondition $\texttt{state}(\texttt{z}_1, \texttt{water})$ is valid in both cases. We thus shift the load of inferring environment-specific actions onto planners and use postconditions for representation, which better captures the semantics of verbs.

Second, because postconditions are higher-level, the number of atoms needed to represent a verb is much less than the corresponding number of actions. For example, the text *"microwave a cup"*, maps to action sequence with 10–15 actions, the postcondition only has two atoms: $\texttt{in}(\texttt{cup}_2, \texttt{microwave}_1) \wedge \texttt{state}(\texttt{microwave}, \texttt{is-on})$. This makes searching for new logical forms more

Table 4.1: Some lexical entries for the verb *turn*

Sentence Context	**Lexical entry** $[turn \Rightarrow (\lambda\vec{v}.S, \xi)]$
"turn on the TV"	$\texttt{state}(\texttt{v}_1, \texttt{is-on}) \wedge \texttt{near}(\texttt{v}_2, \texttt{v}_1)$
	$\xi : \texttt{v}_1 \rightarrow \texttt{tv}_1, \texttt{v}_2 \rightarrow \texttt{robot}_1$
"turn on the right back burner"	$\texttt{state}(\texttt{v}_1, \texttt{fire3}) \wedge \texttt{near}(\texttt{v}_2, \texttt{v}_1)$
	$\xi : \texttt{v}_1 \rightarrow \texttt{stove}_1, \texttt{v}_2 \rightarrow \texttt{robot}_1$
"turn off the water"	$\neg\texttt{state}(\texttt{v}_1, \texttt{tap-on})$
	$\xi : \texttt{v}_1 \rightarrow \texttt{sink}_1$
"turn the television input to xbox"	$\texttt{state}(\texttt{v}_1, \texttt{channel6}) \wedge \texttt{near}(\texttt{v}_1, \texttt{v}_2)$
	$\xi : \texttt{v}_1 \rightarrow \texttt{tv}_1, \texttt{v}_2 \rightarrow \texttt{xbox}_1$

tractable.

Advantages of Anchoring. Similar to the VEIL templates of Misra et al. [2014], the free variables $\vec{v}$ are associated with a mapping ξ to concrete objects. This is useful for resolving ellipsis. Suppose the following lexical entry was created at training time based on the text *"throw the drinks in the trash bag"*:

$$[\ell\text{: } throw \Rightarrow \lambda xyz.S\,(x, y, z)], \text{ where}$$
$$S = \texttt{in}(\texttt{x}, \texttt{y}) \wedge \neg\texttt{grasping}(\texttt{z}, \texttt{x}) \wedge \neg\texttt{state}(\texttt{z}, \texttt{closed})$$
$$\xi = \{\texttt{x} \rightarrow \texttt{coke}_1, \texttt{y} \rightarrow \texttt{garbageBin}_1, \texttt{z} \rightarrow \texttt{robot}_1\}$$

Now consider a new text at test time *"throw away the chips"*, which does not explicitly mention where to throw the chips. Our semantic parsing algorithm (Section 4.5) will use the previous mapping $y \rightarrow \texttt{garbabeBin}_1$ to choose an object most similar to a garbage bin.

4.5 Semantic Parsing Model

Given a sequence of frame nodes $c_{1:k}$ and an initial environment s_1, our semantic parsing model defines a joint distribution over logical forms $z_{1:k}$. Specifically, we define a conditional random field (CRF) over $z_{1:k}$, as shown in Figure 4.2:

$$p_\theta(z_{1:k} \mid c_{1:k}, s_1) \propto \exp\left(\sum_{i=1}^{k} \phi(c_i, z_{i-1}, z_i, s_i) \cdot \theta\right),$$

where $\phi(c_i, z_{i-1}, z_i, e_i)$ is the feature vector and θ is the weight vector. Note that the environments $s_{1:k}$ are a deterministic function of the logical forms $z_{1:k}$ through the recurrence $s_{i+1} = \mathit{simulator}(s_i, \mathit{planner}(s_i, z_i))$, which couples the different time steps.

Features. The feature vector $\phi(c_i, z_{i-1}, z_i, s_i)$ contains 16 features which capture the dependencies between text, logical forms, and environment. Recall that $z_i = ([\nu \Rightarrow (\lambda\vec{v}.S, \xi)], \xi_i)$, where ξ is the environment in which the lexical entry was created and ξ_i is the current environment. Let $f_i = (\lambda\vec{v}.S)(\xi_i)$ be the current postcondition. Here we briefly describe the important features (see the supplemental material for the full list):

- *Language and logical form*: The logical form z_i should generally reference objects mentioned in the text. Assume we have computed a correlation $\rho(\omega, o)$ between each object description ω and object o, whose construction is described later. We then define two features: precision correlation, which encourages z_i to only use objects referred to in c_i; and recall correlation, which encourages z_i to use all the objects referred to in c_i.
- *Logical form*: The postcondition f_i should be based on previously seen environments. For example, microwaving an empty cup and grasping a

couch are unlikely postconditions. We define features corresponding to the average probability (based on the training data) of all conjunctions of at most two atoms in the postcondition (e.g., $\texttt{grasping}(\texttt{robot}, \texttt{cup})\}$). We do the same with their abstract versions ($\{\texttt{grasping}(\texttt{v}_1, \texttt{v}_2)\}$). In addition, we build the same set of four probability tables conditioned on verbs in the training data. For example, the abstract postcondition $\texttt{state}(\texttt{v}_1, \texttt{water})$ has a higher probability conditioned on the verb "*fill*". This gives us a total of 8 features of this type.

- *Logical form and environment*: Recall that anchoring helps us in dealing with ellipsis and noise. We add a feature based on the average correlation between the objects of the new mapping ξ_i with the corresponding objects in the anchored mapping ξ.

The other features are based on the relationship between object descriptions, similarity between ξ and ξ_i and transition probabilities between logical forms z_{i-1} and z_i. These probabilities are also learned from training data.

Mapping Object Descriptions. Our features rely on a mapping from object descriptions ω (e.g., "*the red shiny cup*") to objects o (e.g., $\texttt{cup}_8$), which has been addressed in many recent works [Matuszek et al., 2012a, Guadarrama et al., 2014, Fasola and Mataric, 2014].

One key idea is: instead of computing rigid lexical entries such as *cup* $\rightarrow$ $\texttt{cup}_1$, we use a continuous correlation score $\rho(\omega, o) \in [0, 1]$ that measures how well ω describes o. This flexibility allows the algorithm to use objects not explicitly mentioned in text. Given "*get me a tank of water*", we might choose an approximate vessel (e.g., $\texttt{cup}_2$).

Given an object description ω, an object o, and a set of previously seen objects (used for anaphoric resolution), we define the correlation $\rho(\omega, o)$ using the following approach:

- If ω *is a pronoun*, $\rho(\omega, o)$ is the ratio of the position of the last reference of o to the length of the action sequence computed so far, thus preferring recent objects.
- Otherwise, we compute the correlation using various sources: the object's category; the object's state for handling metonymy (e.g., the description "*coffee*" correlates well with the object $\texttt{mug}_1$ if $\texttt{mug}_1$ contains coffee—$\texttt{state}(\texttt{mug}_1, \texttt{has-coffee})$ is true), WordNet [Fellbaum, 1998] for dealing synonymy and hyponymy; and word alignments between the objects and text from Giza++ [Och and Ney, 2003] to learn domain-specific references (e.g., "*Guinness book*" refers to $\texttt{book}_1$, not $\texttt{book}_2$). More details can be found in the supplemental material.

4.6 Lexicon Induction from Training Data

In order to map text to logical forms, we first induce an initial anchored lexicon Λ from the training data $\{(\bar{x}^{(m)}, s^{(m)}, a^{(m)}, \beta^{(m)})\}_{m=1}^{M}$. At test time, we add new lexical entries (Section 4.7) to Λ.

Recall that shallow parsing $\bar{x}^{(m)}$ yields a list of frame nodes $c_{1:k}$. For each frame node c_i and its aligned action sequence a_i, we take the conjunction of all the atoms (and their negations) which are false in the current one s_i but true in the next environment s_{i+1}. We parametrize this conjunction by replacing each

object with a variable, yielding a postcondition S parametrized by free variables $\vec{v}$ and the mapping ξ from $\vec{v}$ to objects in s_i. We then add the lexical entry $[verb(c_i) \Rightarrow (\lambda\vec{v}.S, \xi)]$ to Λ.

Instantiating Lexical Entries. At test time, for a given clause c_i and environment s_i, we generate set of logical forms $z_i = (\ell_i, \xi_i)$. To do this, we consider the lexical entries in Λ with the same verb as c_i. For each such lexical entry ℓ_i, we can map its free variables $\vec{v}$ to objects in s_i in an exponential number of ways. Therefore, for each ℓ_i we only consider the logical form (ℓ_i, ξ_i) where the mapping ξ_i obtains the highest score under the current model: $\xi_i = \arg\max_{\xi'} \phi(c_i, z_{i-1}, (\ell_i, \xi'), s_i) \cdot \theta$. For the feature vector ϕ that we consider, this approximately translates to solving an integer quadratic program with variables $[y_{ij}] \in \{0, 1\}$, where $y_{ij} = 1$ only if v_i maps to object j.

4.7 Environment-Driven Lexicon Induction at Test Time

Unfortunately, we cannot expect the initial lexicon Λ induced from the training set to have full coverage of the required postconditions. Even after using 90% of the data for training, we encountered 17% new postconditions on the remaining 10%. We therefore propose generating new lexical entries at test time and adding them to Λ.

Formally, for a given environment s_i and frame node c_i, we want to generate likely logical forms. Although the space of all possible logical forms is very large, the environment constrains the possible interpretations. We first compute the set of atoms that are false in s_i and that only contain objects o that are "referred" to by either c_i or c_{i-1}, where "refers" means that there ex-

ists some argument ω in c_i for which $o \in \arg\max_{o'} \rho(\omega, o')$. For example, if c_i corresponds to the text "*distribute pillows among the couches*", we consider the atom $\texttt{on}(\texttt{pillow}_1, \texttt{armchair}_1)$ but not $\texttt{on}(\texttt{pillow}_1, \texttt{snacktable}_2)$ since the object $\texttt{armchair}_1$ has the highest correlation to the description "*couches*".

Next, for each atom, we convert it into a logical form $z = (\ell, \xi)$ by replacing each object with a variable. While this generalization gives us a mapping ξ, we create a lexical entry $\ell_i = [v \Rightarrow (\lambda \vec{v}.S, \emptyset)]$ without it, where S is the parameterized atom. Note that the anchored mapping is empty, representing the fact that this lexical entry was unseen during training time. For example, the atom $\texttt{state}(\texttt{tv}_1, \texttt{mute})$ would be converted to the logical form (ℓ, ξ), where $\ell = [verb(c_i) \Rightarrow (\lambda v.\texttt{state}(v, \texttt{mute}), \emptyset]$ and $\xi = \{v \rightarrow \texttt{tv}_1\}$. We do not generalize state names (e.g., `mute`) because they generally are part of the meaning of the verb.

The score $\phi(c_i, z_{i-1}, z_i, s_i) \cdot \theta$ is computed for the logical form z_i produced by each postcondition. We then take the conjunction of every pair of postconditions corresponding to the 200 highest-scoring logical forms. This gives us new set of postconditions on which we repeat the generalization-scoring-conjunction cycle. We keep doing this while the scores of the new logical forms is increasing or while there are logical forms remaining.

If a logical form $z = ([v \Rightarrow (\lambda \vec{v}.S, \emptyset)], \xi)$ is used by the predicted action sequence, we add the lexical entry $[v \Rightarrow (\lambda \vec{v}.S, \xi)]$ to the lexicon Λ. This is different to other lexicon induction procedures such as GENLEX [Zettlemoyer and Collins, 2007] which are done at training time only and require more supervision. Moreover, GENLEX does not use the environment context in creating new lexical entries and thus is not appropriate at test time, since it would vastly

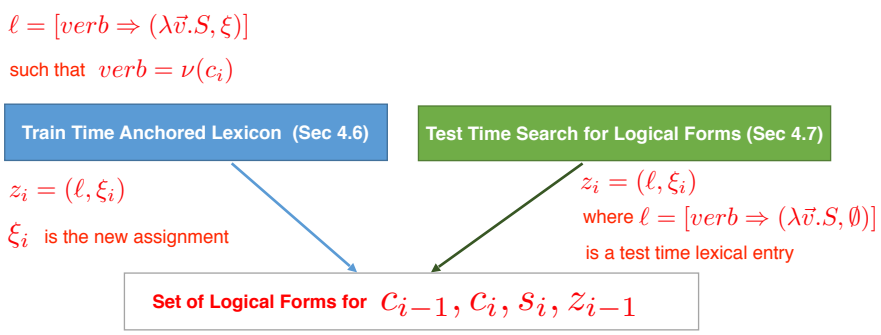


Figure 4.4: Logical forms for given clauses c_{i-1} and c_i, environment s_i, and previous logical form z_{i-1} are generated from both a lexicon induced from training data and a test-time search procedure based on the environment.

overgenerate lexical entries compared to our approach. For us, the environment thus provides implicit supervision for lexicon induction.

4.8 Inference and Parameter Estimation

Inference. Given a text $\bar{x}$ (which is converted to $c_{1:k}$ via Section 4.3.2) and an initial environment s_1, we wish to predict an action sequence a based on $p_\theta(a_{1:k} \mid c_{1:k}, s_1)$, which marginalizes over all logical forms $z_{1:k}$ (see Figure 4.2).

To enumerate possible logical forms, semantic parsers typically lean heavily on a lexicon [Artzi and Zettlemoyer, 2013], leading to high precision but lower recall, or search more aggressively [Berant et al., 2013], leading to higher recall but lower precision. We adopt the following hybrid approach: Given s_i, c_{i-1}, c_i and z_{i-1}, we use both the lexical entries in Λ as explained in Section 4.6 and the search procedure in Section 4.7 to generate the set of possible logical forms for z_i (see Figure 4.4). We use beam search, keeping only the highest-scoring logical form with satisfiable postconditions for each $i \in \{1, \ldots, k\}$ and resulting action sequence $a_{1:i}$.

Table 4.2: New verbs and concepts induced at test time (Section 4.7). Postconditions denote the learned logical form representing the meaning of the underlined verb.

Instruction	*"mix it with ice cream and syrup"*
Learned Postcondition	$\texttt{state}(\texttt{cup}_2, \texttt{ice-cream}_1) \wedge \texttt{state}(\texttt{cup}_2, \texttt{vanilla})$
# Log. forms explored	15
Instruction	*"distribute among the couches"*
Learned Postcondition	$\wedge_{\texttt{j}\in\{1,3\}}\texttt{on}(\texttt{pillow}_\texttt{j}, \texttt{loveseat}_1) \wedge \texttt{on}(\texttt{pillow}_{\texttt{i}+1}, \texttt{armchair}_{\texttt{i}+1})$
# Log. forms explored	386
Instruction	*"boil it on the stove"*
Learned Postcondition	$\texttt{state}(\texttt{stove}, \texttt{stovefire1}) \wedge \texttt{state}(\texttt{kettle}, \texttt{water})$
# Log. forms explored	109
Instruction	*"change the channel to a movie"*
Learned Postcondition	$\texttt{state}(\texttt{tv}_1, \texttt{channel4}) \wedge \texttt{on}(\texttt{book}_1, \texttt{loveseat}_1)$
# Log. forms explored	98

Parameter Estimation. We split 10% of our training data into a separate tuning set (the 90% was used to infer the lexicon). On each example in this set, we extracted the full sequence of logical forms $z_{1:k}$ from the action sequence $a_{1:k}$ based on Section 4.6. For efficiency, we used an objective similar to pseudo-likelihood to estimate the parameters θ. Specifically, we maximize the average log-likelihood over each adjacent pair of logical forms under $\tilde{p}_\theta$:

$$\tilde{p}_\theta(z_i \mid z_{i-1}, c_i, s_i) \propto \exp(\phi(c_i, z_{i-1}, z_i, s_i)^\top \theta).$$

The weights were initialized to 0. We performed 300 iterations over the validation set with a learning rate of $\frac{0.005}{N}$.

4.9 Dataset and Experiments

4.9.1 Dataset

We collected a dataset of 500 examples from 62 people using a crowdsourcing system similar to Misra et al. [2014]. We consider two different 3D scenarios: a kitchen and a living room, each containing an average of 40 objects. Both of these scenarios have 10 environments consisting of different sets of objects in different configurations. We define 10 high-level objectives, 5 per scenario, such as *clean the room*, *make coffee*, *prepare room for movie night*, etc.

One group of users wrote natural language commands to achieve the high-level objectives. Another group controlled a virtual robot to accomplish the commands given by the first group. The dataset contains considerable variety, consisting of 148 different verbs, an average of 48.7 words per text, and an average of 21.5 actions per action sequence. Users make spelling and grammar errors in addition to occasionally taking random actions not relevant to the text. The supplementary material contains more details.

We filtered out 31 examples containing fewer than two action sequences. Of the remaining examples, 378 were used for training and 91 were used for test. Our algorithm is tested on four new environments (two from each scenario).

4.9.2 Experiments and Results

Evaluation Metrics. We consider two metrics, *IED* and *END*, which measure accuracy based on the action sequence and environment, respectively. Specifi-

cally, the *IED* metric [Misra et al., 2014] is the edit distance between predicted and true action sequence. The *END* metric is the Jaccard index of sets A and B, where A is the set of atoms (e.g., $\texttt{on(cup}_1\texttt{,table}_1\texttt{)}$) whose truth value changed due to simulating the predicted action sequence, and B is that of the true action sequence.

Baselines. We compare our algorithm with the following baselines:

1. Chance: Randomly selects a logical form for every frame node from the set of logical forms generated by generalizing all possible postconditions that do not hold in the current environment. These postconditions could contain up to 93 atoms.
2. Manually Defined Templates: Defines a set of postcondition templates for verbs similar to Guadarrama et al. [2013].
3. UBL-Best Parse [Kwiatkowski et al., 2010]: UBL algorithm trained on text aligned with postconditions and a noun-phrase seed lexicon. The planner uses the highest scoring postcondition given by UBL to infer the action sequence.
4. VEIL [Misra et al., 2014]: Uses action sequences as logical forms and does not generate lexical entries at test time.
We also consider two variations of our model: (i) using only lexical entries induced using the training data, and (ii) using only the logical forms induced at test-time by the search procedure.

The results are presented in Table 4.3. We observe that our full model outperforms the baseline and the two pure search- and lexicon-based variations of our model. We further observe that adding the search procedure (Section 4.7) improved the accuracy by 1.5% on IED and 2% on END. The logical forms generated by the search were able to successfully map 48% of the new verbs.

Table 4.3: Results on the metrics and baselines described in section 4.9.2. The numbers are normalized to 100 with larger values being better.

Algorithm	**IED**	**END**
Chance	0.3	0.5
Manually Defined Templates	2.5	1.8
UBL- Best Parse [Kwiatkowski et al., 2010]	5.3	6.9
VEIL [Misra et al., 2014]	14.8	20.7
Model with only train-time lexicon induction	20.8	26.8
Model with only test-time lexicon induction	21.9	25.9
Full Model	**22.3**	**28.8**

Table 4.2 shows new verbs and concepts that the algorithm was able to induce at test time. The algorithm was able to correctly learn the lexical entries for the verbs "*distribute*" and "*mix*", while the ones for verbs "*change*" and "*boil*" were only partly correct. The postconditions in Table 4.2 are not structurally isomorphic to previously-seen logical forms; hence they could not have been handled by using synonyms or factored lexicons [Kwiatkowski et al., 2011]. The poor performance of UBL was because the best logical form often produced an unsatisfiable postcondition. This can be remedied by joint modeling with the environment. The VEIL baseline used actions for representation and does not generalize as well as the postconditions in our logical forms.

It is also instructive to examine the alternate postconditions that the search procedure considers. For the first example in Table 4.2, the following postcondition was considered by not selected:

$$\texttt{grasping}(\texttt{robot}, \texttt{icecream}_2) \wedge \texttt{grasping}(\texttt{robot}, \texttt{syrup}_1).$$

While this postcondition uses all the objects described in the text, the environment-based features suggest it makes little sense for the task to end with the robot eternally grasping objects. For the second example, alternate postconditions considered included:

1. $\texttt{on}(\texttt{pillow}_1, \texttt{pillow}_2) \wedge \texttt{on}(\texttt{pillow}_3, \texttt{pillow}_4)$
2. $\wedge_{j=1}^{4} \texttt{on}(\texttt{pillow}_j, \texttt{loveseat}_1)$
3. $\wedge_{j=1}^{3} \texttt{near}(\texttt{robot}_1, \texttt{armchair}_j)$

The algorithm did not choose options 1 or 3 since the environment-based features recognizes these as unlikely configurations. Option 2 was ruled out since the recall correlation feature realizes that not all the couches are mentioned in the postcondition.

To test how much features on the environment help, we removed all such features from our full model. We found that the accuracy fell to 16.0% on the IED metric and 16.6% on the END metric, showing that the environment is crucial.

In this work, we relied on a simple deterministic shallow parsing step. We found that shallow parsing was able to correctly process the text in only 46% of the test examples, suggesting that improving this initial component or at least modeling the uncertainty there would be beneficial.

4.10 Conclusion

We have presented an algorithm for mapping text to actions that induces lexical entries at test time using the environment. Our algorithm couples the lexicon extracted from training data with a test-time search that uses the environment to reduce the space of logical forms. Our results suggest that using the environment to provide lexical coverage of high-level concepts is a promising avenue for further research.

Reproducibility. Code, data, and experiments for this paper are available on the CodaLab platform at `https://www.codalab.org/worksheets/0x7f9151ec074f4f589e4d4786db7bb6de/`.

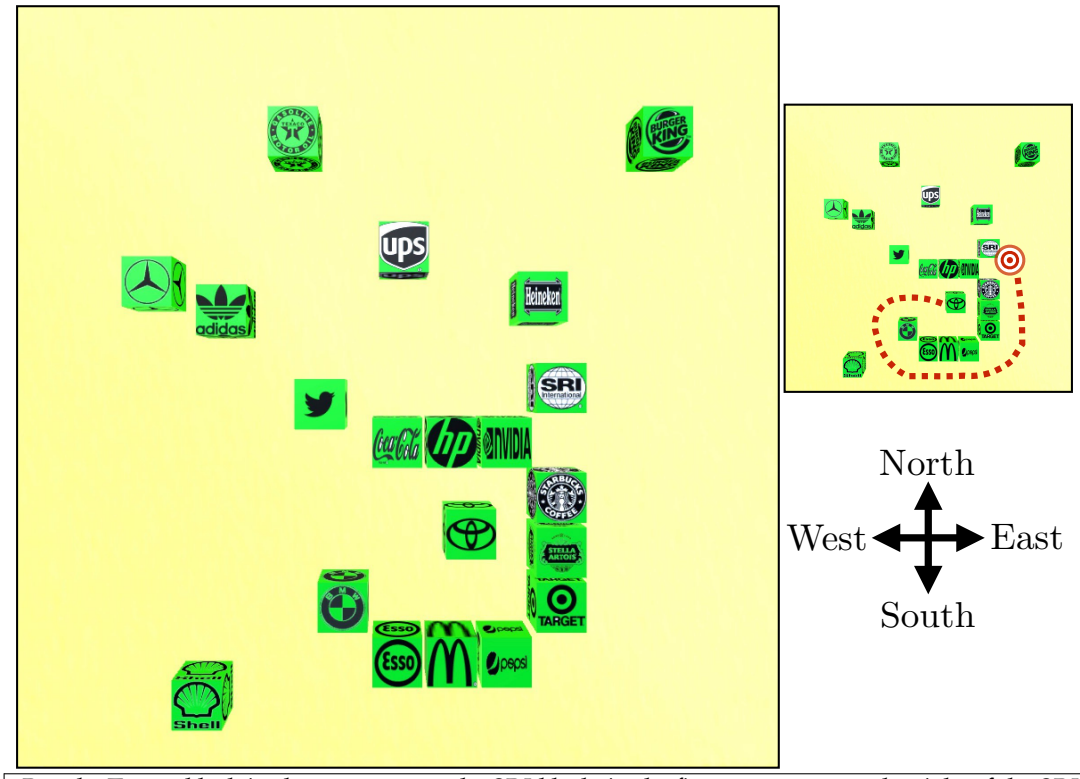
CHAPTER 5

A SINGLE MODEL APPROACH

5.1 Introduction

An agent executing natural language instructions requires robust understanding of language and its environment. Existing approaches addressing this problem assume structured environment representations [Chen and Mooney, 2011, Mei et al., 2016a], or combine separately trained models [Matuszek et al., 2010, Tellex et al., 2011], including for language understanding and visual reasoning. We propose to directly map text and raw image input to actions with a single learned model. This approach offers multiple benefits, such as not requiring intermediate representations, planning procedures, or training multiple models.

Figure 5.1 illustrates the problem in the Blocks environment [Bisk et al., 2016b]. The agent observes the environment as an RGB image using a camera sensor. Given the RGB input, the agent must recognize the blocks and their layout. To understand the instruction, the agent must identify the block to move (Toyota block) and the destination (just right of the SRI block). This requires solving semantic and grounding problems. For example, consider the topmost instruction in the figure. The agent needs to identify the phrase referring to the block to move, *Toyota block*, and ground it. It must resolve and ground the phrase *SRI block* as a reference position, which is then modified by the spatial meaning recovered from *the same row as* or *first open space to the right of*, to identify the goal position. Finally, the agent needs to generate actions, for example moving the Toyota block around obstructing blocks.



Put the Toyota block in the same row as the SRI block, in the first open space to the right of the SRI block
Move Toyota to the immediate right of SRI, evenly aligned and slightly separated
Move the Toyota block around the pile and place it just to the right of the SRI block
Place Toyota block just to the right of The SRI Block
Toyota, right side of SRI

Figure 5.1: Instructions in the Blocks environment. The instructions all describe the same task. Given the observed RGB image of the start state (large image), our goal is to execute such instructions. In this task, the direct-line path to the target position is blocked, and the agent must plan and move the Toyota block around. The small image marks the target and an example path, which includes 34 steps.

To address these challenges with a single model, we design a neural network agent. The agent executes instructions by generating a sequence of actions. At each step, the agent takes as input the instruction text, observes the world as an RGB image, and selects the next action. Action execution changes the state of the world. Given an observation of the new world state, the agent selects the next action. This process continues until the agent indicates execution comple-

tion. When selecting actions, the agent jointly reasons about its observations and the instruction text. This enables decisions based on close interaction between observations and linguistic input.

We train the agent with different levels of supervision, including complete demonstrations of the desired behavior and annotations of the goal state only. While the learning problem can be easily cast as a supervised learning problem, learning only from the states observed in the training data results in poor generalization and failure to recover from test errors. We use reinforcement learning [Sutton and Barto, 2018] to observe a broader set of states through exploration. Following recent work in robotics [Levine et al., 2016, Rusu et al., 2016], we assume the training environment, in contrast to the test environment, is instrumented and provides access to the state. This enables a simple problem reward function that uses the state and provides positive reward on task completion only. This type of reward offers two important advantages: (a) it is a simple way to express the ideal agent behavior we wish to achieve, and (b) it creates a platform to add training data information.

We use reward shaping [Ng et al., 1999] to exploit the training data and add to the reward additional information. The modularity of shaping allows varying the amount of supervision, for example by using complete demonstrations for only a fraction of the training examples. Shaping also naturally associates actions with immediate reward. This enables learning in a contextual bandit setting [Auer et al., 2002b, Langford and Zhang, 2008], where optimizing the immediate reward is sufficient and has better sample complexity than unconstrained reinforcement learning [Agarwal et al., 2014].

We evaluate with the block world environment and data of Bisk et al.

[2016b], where each instruction moves one block (Figure 5.1). While the original task focused on source and target prediction only, we build an interactive simulator and formulate the task of predicting the complete sequence of actions. At each step, the agent must select between 81 actions with 15.4 steps required to complete a task on average, significantly more than existing environments [Chen and Mooney, 2011]. Our experiments demonstrate that our reinforcement learning approach effectively reduces execution error by 24% over standard supervised learning and 34-39% over common reinforcement learning techniques. Our simulator, code, models, and execution videos are available at: `https://github.com/clic-lab/blocks`.

5.2 Technical Overview

Task Let $\mathcal{X}$ be the set of all *instructions*, $\mathcal{S}$ the set of all *world states*, and $\mathcal{A}$ the set of all *actions*. An instruction $\bar{x} \in \mathcal{X}$ is a sequence $\langle x_1, \ldots, x_n \rangle$, where each x_i is a token. The agent executes instructions by generating a sequence of actions, and indicates execution completion with the special action `STOP`. Action execution modifies the world state following a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The execution $\bar{e}$ of an instruction $\bar{x}$ starting from s_1 is an m-length sequence $\langle (s_1, a_1), \ldots, (s_m, a_m) \rangle$, where $s_j \in \mathcal{S}$, $a_j \in \mathcal{A}$, $T(s_j, a_j) = s_{j+1}$ and $a_m =$ `STOP`. In Blocks (Figure 5.1), a state specifies the positions of all blocks. For each action, the agent moves a single block on the plane in one of four directions (north, south, east, or west). There are 20 blocks, and 81 possible actions at each step, including `STOP`. For example, to correctly execute the instructions in the figure, the agent's likely first action is `TOYOTA-WEST`, which moves the Toyota block one step west. Blocks can not move over or through other blocks.

Model The agent observes the world state via a visual sensor (i.e., a camera). Given a world state s, the agent observes an RGB image I generated by the function $\text{IMG}(s)$. We distinguish between the world state s and the *agent context*[1] $\tilde{s}$, which includes the instruction, the observed image $\text{IMG}(s)$, images of previous states, and the previous action. To map instructions to actions, the agent reasons about the agent context $\tilde{s}$ to generate a sequence of actions. At each step, the agent generates a single action. We model the agent with a neural network policy. At each step j, the network takes as input the current agent context $\tilde{s}_j$, and predicts the next action to execute a_j. We formally define the agent context and model in Section 5.3.

Learning We assume access to training data with N examples $\{(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})\}_{i=1}^{N}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $\bar{e}^{(i)}$ is an execution demonstration of $\bar{x}^{(i)}$ starting at $s_1^{(i)}$. We use policy gradient (Section 5.4) with reward shaping derived from the training data to increase learning speed and exploration effectiveness (Section 5.5). Following work in robotics [Levine et al., 2016], we assume an instrumented environment with access to the world state to compute the reward during training only. We define our approach in general terms with demonstrations, but also experiment with training using goal states.

Evaluation We evaluate task completion error on a test set $\{(\bar{x}^{(i)}, s_1^{(i)}, s_g^{(i)})\}_{i=1}^{M}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $s_g^{(i)}$ is the goal state. We measure execution error as the distance between the final execution state and $s_g^{(i)}$.

[1] We use the term *context* similar to how it is used in the contextual bandit literature to refer to the information available for decision making. While agent contexts capture information about the world state, they do not include physical information, except as captured by observed images.

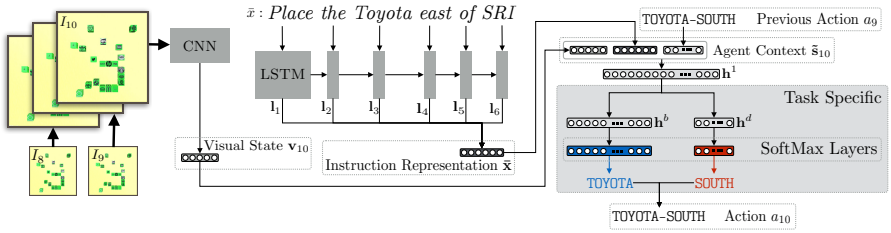


Figure 5.2: Illustration of the policy architecture showing the 10th step in the execution of the instruction *Place the Toyota east of SRI* in the state from Figure 5.1. The network takes as input the instruction $\bar{x}$, image of the current state I_{10}, images of previous states I_8 and I_9 (with $K = 2$), and the previous action a_9. The text and images are embedded with LSTM and CNN. The actions are selected with the task specific multi-layer perceptron.

5.3 Model

We model the agent policy π with a neural network. The agent observes the instruction and an RGB image of the world. Given a world state s, the image I is generated using the function IMG(s). The instruction execution is generated one step at a time. At each step j, the agent observes an image I_j of the current world state s_j and the instruction $\bar{x}$, predicts the action a_j, and executes it to transition to the next state s_{j+1}. This process continues until `STOP` is predicted and the agent stops, indicating instruction completion. The agent also has access to K images of previous states and the previous action to distinguish between different stages of the execution [Mnih et al., 2015]. Figure 5.2 illustrates our architecture.

Formally,[2] at step j, the agent considers an agent context $\tilde{s}_j$, which is a tuple $(\bar{x}, I_j, I_{j-1}, \ldots, I_{j-K}, a_{j-1})$, where $\bar{x}$ is the natural language instruction, I_j is an image of the current world state, the images $I_{j-1}, \ldots, I_{j-K}$ represent K previous states, and a_{j-1} is the previous action. The agent context includes information

[2]We use bold-face capital letters for matrices and bold-face lowercase letters for vectors. Computed input and state representations use bold versions of the symbols. For example, $\bar{\mathbf{x}}$ is the computed representation of an instruction $\bar{x}$.

about the current state and the execution. Considering the previous action a_{j-1} allows the agent to avoid repeating failed actions, for example when trying to move in the direction of an obstacle. In Figure 5.2, the agent is given the instruction *Place the Toyota east of SRI*, is at the 10-th execution step, and considers $K = 2$ previous images.

We generate continuous vector representations for all inputs, and jointly reason about both text and image modalities to select the next action. We use a recurrent neural network (RNN) [Elman, 1990] with a long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] recurrence to map the instruction $\bar{x} = \langle x_1, \ldots, x_n \rangle$ to a vector representation $\bar{\mathbf{x}}$. Each token x_i is mapped to a fixed dimensional vector with the learned embedding function $\psi(x_i)$. The instruction representation $\bar{\mathbf{x}}$ is computed by applying the LSTM recurrence to generate a sequence of hidden states $\mathbf{l}_i = \text{LSTM}(\psi(x_i), \mathbf{l}_{i-1})$, and computing the mean $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{l}_i$ [Narasimhan et al., 2015]. The current image I_j and previous images $I_{j-1}, \ldots, I_{j-K}$ are concatenated along the channel dimension and embedded with a convolutional neural network (CNN) to generate the visual state $\mathbf{v}$ [Mnih et al., 2013]. The last action a_{j-1} is embedded with the function $\psi_a(a_{j-1})$. The vectors $\mathbf{v}_j$, $\bar{\mathbf{x}}$, and $\psi_a(a_{j-1})$ are concatenated to create the agent context vector representation $\tilde{\mathbf{s}}_j = [\mathbf{v}_j, \bar{\mathbf{x}}, \psi_a(a_{j-1})]$.

To compute the action to execute, we use a feed-forward perceptron that decomposes according to the domain actions. This computation selects the next action conditioned on the instruction text and observations from both the current world state and recent history. In the block world domain, where actions decompose to selecting the block to move and the direction, the network computes block and direction probabilities. Formally, we decompose an action a to

direction a^D and block a^B. We compute the feedforward network:

$$\begin{aligned}
\mathbf{h}^1 &= \max(\mathbf{W}^{(1)}\tilde{\mathbf{s}}_j + \mathbf{b}^{(1)}, 0) \\
\mathbf{h}^D &= \mathbf{W}^{(D)}\mathbf{h}^1 + \mathbf{b}^{(D)} \\
\mathbf{h}^B &= \mathbf{W}^{(B)}\mathbf{h}^1 + \mathbf{b}^{(B)} \quad ,
\end{aligned}$$

and the action probability is a product of the component probabilities:

$$\begin{aligned}
P(a_j^D = d \mid \bar{x}, s_j, a_{j-1}) &\propto \exp(\mathbf{h}_d^D) \\
P(a_j^B = b \mid \bar{x}, s_j, a_{j-1}) &\propto \exp(\mathbf{h}_b^B) \quad .
\end{aligned}$$

At the beginning of execution, the first action a_0 is set to the special value `NONE`, and previous images are zero matrices. The embedding function ψ is a learned matrix. The function ψ_a concatenates the embeddings of a_{j-1}^D and a_{j-1}^B, which are obtained from learned matrices, to compute the embedding of a_{j-1}. The model parameters θ include $\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(D)}$, $\mathbf{b}^{(D)}$, $\mathbf{W}^{(B)}$, $\mathbf{b}^{(B)}$, the parameters of the LSTM recurrence, the parameters of the convolutional network CNN, and the embedding matrices. In our experiments (Section 5.6), all parameters are learned without external resources.

5.4 Learning

We use policy gradient for reinforcement learning [Williams, 1992] to estimate the parameters θ of the agent policy. We assume access to a training set of N examples $\{(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})\}_{i=1}^N$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $\bar{e}^{(i)}$ is an execution demonstration starting from $s_1^{(i)}$ of instruction $\bar{x}^{(i)}$. The main learning challenge is learning how to execute instructions given raw visual input from

relatively limited data. We learn in a contextual bandit setting, which provides theoretical advantages over general reinforcement learning. In Section 5.7, we verify this empirically.

Reward Function The instruction execution problem defines a simple problem reward to measure task completion. The agent receives a positive reward when the task is completed, a negative reward for incorrect completion (i.e., `STOP` in the wrong state) and actions that fail to execute (e.g., when the direction is blocked), and a small penalty otherwise, which induces a preference for shorter trajectories. To compute the reward, we assume access to the world state. This learning setup is inspired by work in robotics, where it is achieved by instrumenting the training environment [Schulman et al., 2015a, Rusu et al., 2016, Levine et al., 2016]. The agent, on the other hand, only uses the agent context (Section 5.3). When deployed, the system relies on visual observations and natural language instructions only. The reward function $R^{(i)} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined for each training example $(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})$, $i = 1 \ldots N$:

$$R^{(i)}(s, a) = \begin{cases} 1.0 & \text{if } s = s_{m^{(i)}} \text{ and } a = \texttt{STOP} \\ -1.0 & s \neq s_{m^{(i)}} \text{ and } a = \texttt{STOP} \\ -1.0 & a \text{ fails to execute} \\ -\delta & \text{else} \end{cases},$$

where $m^{(i)}$ is the length of $\bar{e}^{(i)}$.

The reward function does not provide intermediate positive feedback to the agent for actions that bring it closer to its goal. When the agent explores randomly early during learning, it is unlikely to encounter the goal state due to

the large number of steps required to execute tasks. As a result, the agent does not observe positive reward and fails to learn. In Section 5.5, we describe how reward shaping, a method to augment the reward with additional information, is used to take advantage of the training data and address this challenge.

Policy Gradient Objective We adapt the policy gradient objective defined by Sutton et al. [1999] to multiple starting states and reward functions:

$$\mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} V_{\pi}^{(i)}(s_1^{(i)}) \ ,$$

where $V_{\pi}^{(i)}(s_1^{(i)})$ is the value given by $R^{(i)}$ starting from $s_1^{(i)}$ under the policy π. The summation expresses the goal of learning a behavior parameterized by natural language instructions.

Contextual Bandit Setting In contrast to most policy gradient approaches, we apply the objective to a contextual bandit setting where immediate reward is optimized rather than total expected reward. The primary theoretical advantage of contextual bandits is much tighter sample complexity bounds when comparing upper bounds for contextual bandits [Langford and Zhang, 2008] even with an adversarial sequence of contexts [Auer et al., 2002b] to lower bounds [Krishnamurthy et al., 2016b] or upper bounds [Kearns et al., 1999] for total reward maximization. This property is particularly suitable for the few-sample regime common in natural language problems. While reinforcement learning with neural network policies is known to require large amounts of training data [Mnih et al., 2015], the limited number of training sentences constrains the diversity and volume of agent contexts we can observe during training. Empirically, this

translates to poor results when optimizing the total reward (REINFORCE baseline in Section 5.7). To derive the approximate gradient, we use the likelihood ratio method:

$$\nabla_\theta \mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}[\nabla_\theta \log \pi(\tilde{s}, a) R^{(i)}(s, a)] \quad ,$$

where reward is computed from the world state but policy is learned on the agent context. We approximate the gradient using sampling.

This training regime, where immediate reward optimization is sufficient to optimize policy parameters θ, is enabled by the shaped reward we introduce in Section 5.5. While the objective is designed to work best with the shaped reward, the algorithm remains the same for any choice of reward definition including the original problem reward or several possibilities formed by reward shaping.

Entropy Penalty We observe that early in training, the agent is overwhelmed with negative reward and rarely completes the task. This results in the policy π rapidly converging towards a suboptimal deterministic policy with an entropy of 0. To delay premature convergence we add an entropy term to the objective [Williams and Peng, 1991, Mnih et al., 2016]. The entropy term encourages a uniform distribution policy, and in practice stimulates exploration early during training. The regularized gradient is:

$$\nabla_\theta \mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}[\nabla_\theta \log \pi(\tilde{s}, a) R^{(i)}(s, a) + \lambda \nabla_\theta H(\pi(\tilde{s}, \cdot))] \quad ,$$

Algorithm 1 Policy gradient learning

Input: Training set $\{(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})\}_{i=1}^{N}$, learning rate μ, epochs T, horizon J, and entropy regularization term λ.

Definitions: IMG(s) is a camera sensor that reports an RGB image of state s. π is a probabilistic neural network policy parameterized by θ, as described in Section 5.3. EXECUTE(s, a) executes the action a at the state s, and returns the new state. $R^{(i)}$ is the reward function for example i. ADAM(Δ) applies a per-feature learning rate to the gradient Δ [Kingma and Ba, 2014].

Output: Policy parameters θ.

1: » Iterate over the training data.
2: **for** $t = 1$ to T, $i = 1$ to N **do**
3: $\quad I_{1-K}, \ldots, I_0 = \vec{0}$
4: $\quad a_0 =$ `NONE`, $s_1 = s_1^{(i)}$
5: $\quad j = 1$
6: $\quad$ » Rollout up to episode limit.
7: $\quad$ **while** $j \leq J$ and $a_j \neq$ `STOP` **do**
8: $\quad\quad$ » Observe world and construct agent context.
9: $\quad\quad I_j = \text{IMG}(s_j)$
10: $\quad\quad \tilde{s}_j = (\bar{x}^{(i)}, I_j, I_{j-1}, \ldots, I_{j-K}, a_{j-1}^{d})$
11: $\quad\quad$ » Sample an action from the policy.
12: $\quad\quad a_j \sim \pi(\tilde{s}_j, a)$
13: $\quad\quad s_{j+1} = \text{EXECUTE}(s_j, a_j)$
14: $\quad\quad$ » Compute the approximate gradient.
15: $\quad\quad \Delta_j \leftarrow \nabla_\theta \log \pi(\tilde{s}_j, a_j) R^{(i)}(s_j, a_j) + \lambda \nabla_\theta H(\pi(\tilde{s}_j, \cdot))$
16: $\quad\quad j\mathrel{+}= 1$
17: $\quad \theta \leftarrow \theta + \mu \text{ADAM}(\frac{1}{j} \sum_{j'=1}^{j} \Delta_{j'})$
18: **return** θ

where $H(\pi(\tilde{s}, \cdot))$ is the entropy of π given the agent context $\tilde{s}$, λ is a hyperparameter that controls the strength of the regularization. While the entropy term delays premature convergence, it does not eliminate it. Similar issues are observed for vanilla policy gradient [Mnih et al., 2016].

Algorithm Algorithm 1 shows our learning algorithm. We iterate over the data T times. In each epoch, for each training example $(\bar{x}^{(i)}, s_1^{(i)}, \bar{e}^{(i)})$, $i = 1 \ldots N$, we perform a rollout using our policy to generate an execution (lines 7 - 16). The length of the rollout is bound by J, but may be shorter if the agent selected the `STOP` action. At each step j, the agent updates the agent context $\tilde{s}_j$ (lines 9 - 10),

samples an action from the policy π (line 12), and executes it to generate the new world state s_{j+1} (line 13). The gradient is approximated using the sampled action with the computed reward $R^{(i)}(s_j, a_j)$ (line 15). Following each rollout, we update the parameters θ with the mean of the gradients using ADAM [Kingma and Ba, 2014].

5.5 Reward Shaping

Reward shaping is a method for transforming a reward function by adding a *shaping term* to the problem reward. The goal is to generate more informative updates by adding information to the reward. We use this method to leverage the training demonstrations, a common form of supervision for training systems that map language to actions. Reward shaping allows us to fully use this type of supervision in a reinforcement learning framework, and effectively combine learning from demonstrations and exploration.

Adding an arbitrary shaping term can change the optimality of policies and modify the original problem, for example by making bad policies according to the problem reward optimal according to the shaped function.[3] Ng et al. [1999] and Wiewiora et al. [2003] outline potential-based terms that realize sufficient conditions for *safe* shaping.[4] Adding a shaping term is safe if the order of policies according to the shaped reward is identical to the order according to the original problem reward. While safe shaping only applies to optimizing the total reward, we show empirically the effectiveness of the safe shaping terms we

[3] For example, adding a shaping term $F = -R$ will result in a shaped reward that is always 0, and any policy will be trivially optimal with respect to it.

[4] For convenience, we briefly overview the theorems of Ng et al. [1999] and Wiewiora et al. [2003] in Appendix B.1.

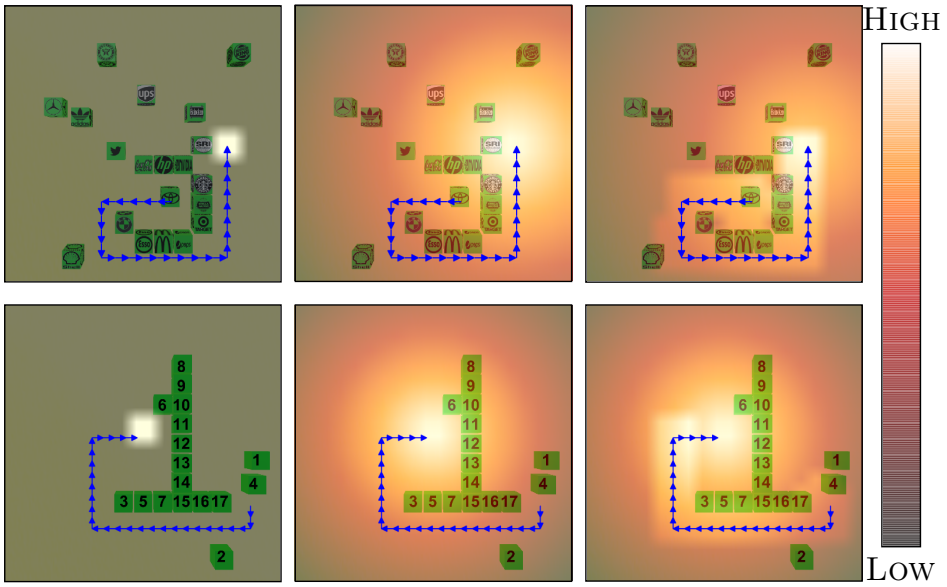


Figure 5.3: Visualization of the shaping potentials for two tasks. We show demonstrations (blue arrows), but omit instructions. To visualize the potentials intensity, we assume only the target block can be moved, while rewards and potentials are computed for any block movement. We illustrate the sparse problem reward (left column) as a potential function and consider only its positive component, which is focused on the goal. The middle column adds the distance-based potential. The right adds both potentials.

design in a contextual bandit setting.

We introduce two shaping terms. The final shaped reward is a sum of them and the problem reward. Similar to the problem reward, we define example-specific shaping terms. We modify the reward function signature as required.

Distance-based Shaping (F_1) The first shaping term measures if the agent moved closer to the goal state. We design it to be a safe potential-based term [Ng et al., 1999]:

$$F_1^{(i)}(s_j, a_j, s_{j+1}) = \phi_1^{(i)}(s_{j+1}) - \phi_1^{(i)}(s_j) \quad .$$

The potential $\phi_1^{(i)}(s)$ is proportional to the negative distance from the goal state $s_g^{(i)}$. Formally, $\phi_1^{(i)}(s) = -\eta \| s - s_g^{(i)} \|$, where η is a constant scaling factor, and $\|.\|$ is a

distance metric. In the block world, the distance between two states is the sum of the Euclidean distances between the positions of each block in the two states, and η is the inverse of block width. The middle column in Figure 5.3 visualizes the potential $\phi_1^{(i)}$.

Trajectory-based Shaping (F_2) Distance-based shaping may lead the agent to sub-optimal states, for example when an obstacle blocks the direct path to the goal state, and the agent must temporarily increase its distance from the goal to bypass it. We incorporate complete trajectories by using a simplification of the shaping term introduced by Brys et al. [2015]. Unlike F_1, it requires access to the previous state and action. It is based on the look-back advice shaping term of Wiewiora et al. [2003], who introduced safe potential-based shaping that considers the previous state and action. The second term is:

$$F_2^{(i)}(s_{j-1}, a_{j-1}, s_j, a_j) = \phi_2^{(i)}(s_j, a_j) - \phi_2^{(i)}(s_{j-1}, a_{j-1}) \ .$$

Given $\bar{e}^{(i)} = \langle (s_1, a_1), \ldots, (s_m, a_m) \rangle$, to compute the potential $\phi_2^{(i)}(s, a)$, we identify the closest state s_j in $\bar{e}^{(i)}$ to s. If $\eta \| s_j - s \| < 1$ and $a_j = a$, $\phi_2^{(i)}(s, a) = 1.0$, else $\phi_2^{(i)}(s, a) = -\delta_f$, where δ_f is a penalty parameter. We use the same distance computation and parameter η as in F_1. When the agent is in a state close to a demonstration state, this term encourages taking the action taken in the related demonstration state. The right column in Figure 5.3 visualizes the effect of the potential $\phi_2^{(i)}$.

5.6 Experimental Setup

Environment We use the environment of Bisk et al. [2016b]. The original task required predicting the source and target positions for a single block given an instruction. In contrast, we address the task of moving blocks on the plane to execute instructions given visual input. This requires generating the complete sequence of actions needed to complete the instruction. The environment contains up to 20 blocks marked with logos or digits. Each block can be moved in four directions. Including the STOP action, in each step, the agent selects between 81 actions. The set of actions is constant and is not limited to the blocks present. The transition function is deterministic. The size of each block step is 0.04 of the board size. The agent observes the board from above. We adopt a relatively challenging setup with a large action space. While a simpler setup, for example decomposing the problem to source and target prediction and using a planner, is likely to perform better, we aim to minimize task-specific assumptions and engineering of separate modules. However, to better understand the problem, we also report results for the decomposed task with a planner.

Data Bisk et al. [2016b] collected a corpus of instructions paired with start and goal states. Figure 5.1 shows example instructions. The original data includes instructions for moving one block or multiple blocks. Single-block instructions are relatively similar to navigation instructions and referring expressions. While they present much of the complexity of natural language understanding and grounding, they rarely display the planning complexity of multi-block instructions, which are beyond the scope of this paper. Furthermore, the original data does not include demonstrations. While generating demonstrations for moving

a single block is straightforward, disambiguating action ordering when multiple blocks are moved is challenging. Therefore, we focus on instructions where a single block changes its position between the start and goal states, and restrict demonstration generation to move the changed block. The remaining data, and the complexity it introduces, provide an important direction for future work.

To create demonstrations, we compute the shortest paths. While this process may introduce noise for instructions that specify specific trajectories (e.g., *move SRI two steps north and ...*) rather than only describing the goal state, analysis of the data shows this issue is limited. Out of 100 sampled instructions, 92 describe the goal state rather than the trajectory. A secondary source of noise is due to discretization of the state space. As a result, the agent often can not reach the exact target position. The demonstrations error illustrates this problem (Table 5.3). To provide task completion reward during learning, we relax the state comparison, and consider states to be equal if the sum of block distances is under the size of one block.

The corpus includes 11,871/1,719/3,177 instructions for training, development and testing. Table 5.1 shows corpus statistic compared to the commonly used SAIL navigation corpus [MacMahon et al., 2006, Chen and Mooney, 2011]. While the SAIL agent only observes its immediate surroundings, overall the blocks domain provides more complex instructions. Furthermore, the SAIL environment includes only 400 states, which is insufficient for generalization with vision input. We compare to other data sets in Appendix B.4.

Evaluation We evaluate task completion error as the sum of Euclidean distances for each block between its position at the end of the execution and in

	SAIL	Blocks
Number of instructions	3,237	16,767
Mean instruction length	7.96	15.27
Vocabulary	563	1,426
Mean trajectory length	3.12	15.4

Table 5.1: Corpus statistics for the block environment we use and the SAIL navigation domain.

the gold goal state. We divide distances by block size to normalize for the image size. In contrast, Bisk et al. [2016b] evaluate the selection of the source and target positions independently.

Systems We report performance of ablations, the upper bound of following the demonstrations (Demonstrations), and five baselines: (a) STOP: the agent immediately stops, (b) RANDOM: the agent takes random actions, (c) SUPERVISED: supervised learning with maximum-likelihood estimate using demonstration state-action pairs, (d) DQN: deep Q-learning with both shaping terms [Mnih et al., 2015], and (e) REINFORCE: policy gradient with cumulative episodic reward with both shaping terms [Sutton et al., 1999]. Full system details are given in Appendix B.2.

Parameters and Initialization Full details are in Appendix B.3. We consider $K = 4$ previous images, and horizon length $J = 40$. We initialize our model with the SUPERVISED model.

Algorithm	Distance Error		Min. Distance	
	Mean	Med.	Mean	Med.
Demonstrations	0.35	0.30	0.35	0.30
Baselines				
STOP	5.95	5.71	5.95	5.71
RANDOM	15.3	15.70	5.92	5.70
SUPERVISED	4.65	4.45	3.72	3.26
REINFORCE	5.57	5.29	4.50	4.25
DQN	6.04	5.78	5.63	5.49
Our Approach	3.60	3.09	2.72	2.21
w/o Sup. Init	3.78	3.13	2.79	2.21
w/o Prev. Action	3.95	3.44	3.20	2.56
w/o F_1	4.33	3.74	3.29	2.64
w/o F_2	3.74	3.11	3.13	2.49
w/ Distance Reward	8.36	7.82	5.91	5.70
Ensembles				
SUPERVISED	4.64	4.27	3.69	3.22
REINFORCE	5.28	5.23	4.75	4.67
DQN	5.85	5.59	5.60	5.46
Our Approach	**3.59**	**3.03**	**2.63**	**2.15**

Table 5.2: Mean and median (Med.) development results.

Algorithm	Distance Error		Min. Distance	
	Mean	Med.	Mean	Med.
Demonstrations	0.37	0.31	0.37	0.31
STOP	6.23	6.12	6.23	6.12
RANDOM	15.11	15.35	6.21	6.09
Ensembles				
SUPERVISED	4.95	4.53	3.82	3.33
REINFORCE	5.69	5.57	5.11	4.99
DQN	6.15	5.97	5.86	5.77
Our Approach	**3.78**	**3.14**	**2.83**	**2.07**

Table 5.3: Mean and median (Med.) test results.

5.7 Results

Table 5.2 shows development results. We run each experiment three times and report the best result. The RANDOM and STOP baselines illustrate the task complexity of the task. Our approach, including both shaping terms in a contextual bandit setting, significantly outperforms the other methods. SUPERVISED learning demonstrates lower performance. A likely explanation is test-time ex-

ecution errors leading to unfamiliar states with poor later performance [Kakade and Langford, 2002], a form of the covariate shift problem. The low performance of REINFORCE and DQN illustrates the challenge of general reinforcement learning with limited data due to relatively high sample complexity [Kearns et al., 1999, Krishnamurthy et al., 2016b]. We also report results using ensembles of the three models.

We ablate different parts of our approach. Ablations of supervised initialization (our approach w/o sup. init) or the previous action (our approach w/o prev. action) result in increase in error. While the contribution of initialization is modest, it provides faster learning. On average, after two epochs, we observe an error of 3.94 with initialization and 6.01 without. We hypothesize that the F_2 shaping term, which uses full demonstrations, helps to narrow the gap at the end of learning. Without supervised initialization and F_2, the error increases to 5.45 (the 0% point in Figure 5.4). We observe the contribution of each shaping term and their combination. To study the benefit of potential-based shaping, we experiment with a negative distance-to-goal reward. This reward replaces the problem reward and encourages getting closer to the goal (our approach w/distance reward). With this reward, learning fails to converge, leading to a relatively high error.

Figure 5.4 shows our approach with varying amount of supervision. We remove demonstrations from both supervised initialization and the F_2 shaping term. For example, when only 25% are available, only 25% of the data is available for initialization and the F_2 term is only present for this part of the data. While some demonstrations are necessary for effective learning, we get most of the benefit with only 12.5%.

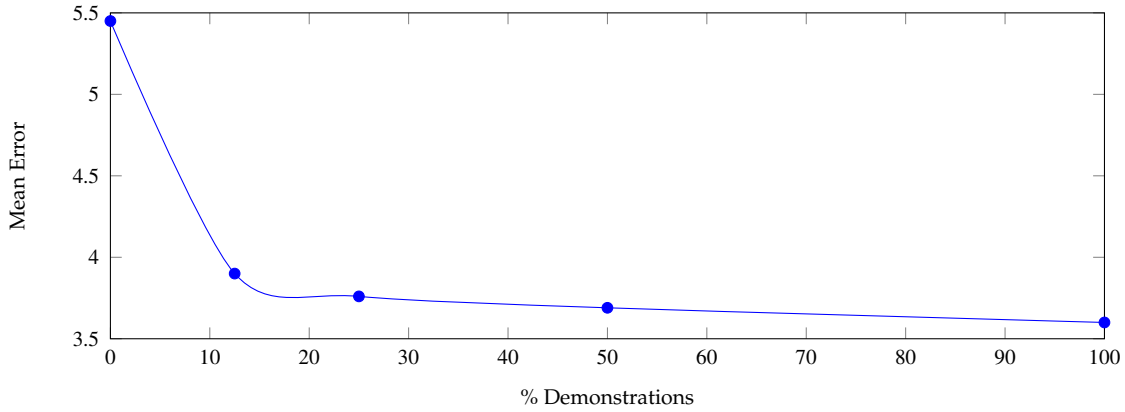


Figure 5.4: Mean distance error as a function of the ratio of training examples that include complete trajectories. The rest of the data includes the goal state only.

Table 5.3 provides test results, using the ensembles to decrease the risk of overfitting the development. We observe similar trends to development result with our approach outperforming all baselines. The remaining gap to the demonstrations upper bound illustrates the need for future work.

To understand performance better, we measure *minimal distance* (min. distance in Tables 5.2 and 5.3), the closest the agent got to the goal. We observe a strong trend: the agent often gets close to the goal and fails to stop. This behavior is also reflected in the number of steps the agent takes. While the mean number of steps in development demonstrations is 15.2, the agent generates on average 28.7 steps, and 55.2% of the time it takes the maximum number of allowed steps (40). Testing on the training data shows an average 21.75 steps and exhausts the number of steps 29.3% of the time. The mean number of steps in training demonstrations is 15.5. This illustrates the challenge of learning how to be behave at an absorbing state, which is observed relatively rarely during training. This behavior also shows in our video.[5]

[5] `https://github.com/clic-lab/blocks`

We also evaluate a supervised learning variant that assumes a perfect planner.[6] This setup is similar to Bisk et al. [2016b], except using raw image input. It allows us to roughly understand how well the agent generates actions. We observe a mean error of 2.78 on the development set, an improvement of almost two points over supervised learning with our approach. This illustrates the complexity of the complete problem.

We conduct a shallow linguistic analysis to understand the agent behavior with regard to differences in the language input. As expected, the agent is sensitive to unknown words. For instructions without unknown words, the mean development error is 3.49. It increases to 3.97 for instructions with a single unknown word, and to 4.19 for two.[7] We also study the agent behavior when observing new phrases composed of known words by looking at instructions with new n-grams and no unknown words. We observe no significant correlation between performance and new bi-grams and tri-grams. We also see no meaningful correlation between instruction length and performance. Although counterintuitive given the linguistic complexities of longer instructions, it aligns with results in machine translation [Luong et al., 2015].

5.8 Conclusions

We study the problem of learning to execute instructions in a situated environment given only raw visual observations. Supervised approaches do not explore adequately to handle test time errors, and reinforcement learning ap-

[6] As there is no sequence of decisions, our reinforcement approach is not appropriate for the planner experiment. The architecture details are described in Appendix B.2.

[7] This trend continues, although the number of instructions is too low (< 20) to be reliable.

proaches require a large number of samples for good convergence. Our solution provides an effective combination of both approaches: reward shaping to create relatively stable optimization in a contextual bandit setting, which takes advantage of a signal similar to supervised learning, with a reinforcement basis that admits substantial exploration and easy avenues for smart initialization. This combination is designed for a few-samples regime, as we address. When the number of samples is unbounded, the drawbacks observed in this scenario for optimizing longer term reward do not hold.

CHAPTER 6

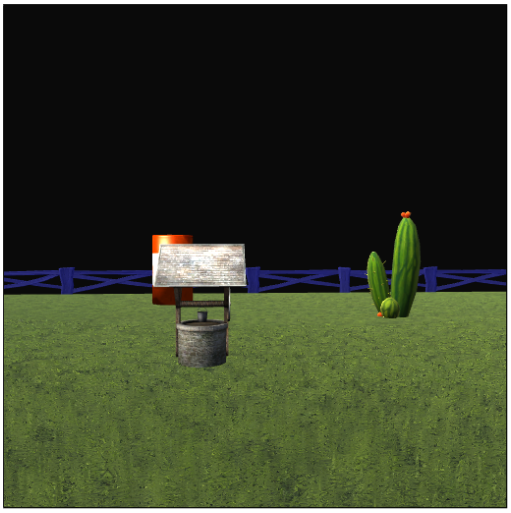
AN INTERPRETABLE MODEL FOR INSTRUCTION FOLLOWING

6.1 Introduction

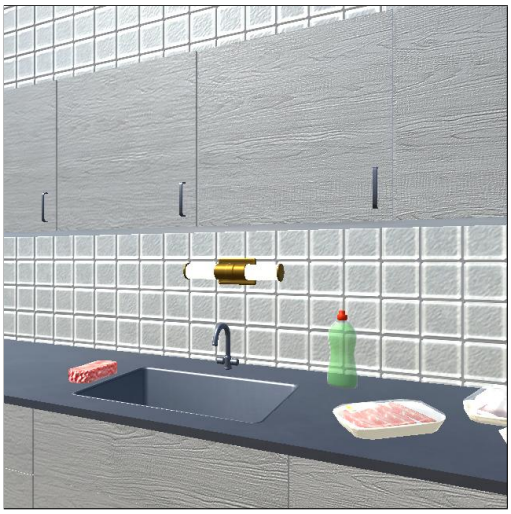
Executing instructions in interactive environments requires mapping natural language and observations to actions. Recent approaches propose learning to directly map from inputs to actions, for example given language and either structured observations [Mei et al., 2016a, Suhr and Artzi, 2018] or raw visual observations [Misra et al., 2017, Xiong et al., 2018]. Rather than using a combination of models, these approaches learn a single model to solve language, perception, and planning challenges. This reduces the amount of engineering required and eliminates the need for hand-crafted meaning representations. At each step, the agent maps its current inputs to the next action using a single learned function that is executed repeatedly until task completion.

Although executing the same computation at each step simplifies modeling, it exemplifies certain inefficiencies; while the agent needs to decide what action to take at each step, identifying its goal is only required once every several steps or even once per execution. The left instruction in Figure 6.1 illustrates this. The agent can compute its goal once given the initial observation, and given this goal can then generate the actions required. In this paper, we study a new model that explicitly distinguishes between goal selection and action generation, and introduce two instruction following benchmark tasks to evaluate it.

Our model decomposes into goal prediction and action generation. Given a natural language instruction and system observations, the model predicts the



After reaching the hydrant head towards the blue fence and pass towards the right side of the well.



Put the cereal, the sponge, and the dishwashing soap into the cupboard above the sink.

Figure 6.1: Example instructions from our two tasks: LANI (left) and CHAI (right). LANI is a landmark navigation task, and CHAI is a corpus of instructions in the CHALET environment.

goal to complete. Given the goal, the model generates a sequence of actions.

The key challenge we address is designing the goal representation. We avoid manually designing a meaning representation, and predict the goal in the agent's observation space. Given the image of the environment the agent observes, we generate a probability distribution over the image to highlight the goal location. We treat this prediction as image generation, and develop LINGUNET, a language conditioned variant of the UNETimage-to-image architecture [Ronneberger et al., 2015]. Given the visual goal prediction, we generate actions using a recurrent neural network (RNN).

Our model decomposition offers two key advantages. First, we can use different learning methods as appropriate for the goal prediction and action generation problems. We find supervised learning more effective for goal prediction, where only a limited amount of natural language data is available. For action

generation, where exploration is critical, we use policy gradient in a contextual bandit setting [Misra et al., 2017]. Second, the goal distribution is easily interpretable by overlaying it on the agent observations. This can be used to increase the safety of physical systems by letting the user verify the goal before any action is executed. Despite the decomposition, our approach retains the advantages of the single-model approach. It does not require designing intermediate representations, and training does not rely on external resources, such as pre-trained parsers or object detectors, instead using demonstrations only.

We introduce two new benchmark tasks with different levels of complexity of goal prediction and action generation. LANI is a 3D navigation environment and corpus, where an agent navigates between landmarks. The corpus includes 6,000 sequences of natural language instructions, each containing on average 4.7 instructions. CHAI is a corpus of 1,596 instruction sequences, each including 7.7 instructions on average, for CHALET, a 3D house environment [Yan et al., 2018]. Instructions combine navigation and simple manipulation, including moving objects and opening containers. Both tasks require solving language challenges, including spatial and temporal reasoning, as well as complex perception and planning problems. While LANI provides a task where most instructions include a single goal, the CHAI instructions often require multiple intermediate goals. For example, the household instruction in Figure 6.1 can be decomposed to eight goals: opening the cupboard, picking each item and moving it to the cupboard, and closing the cupboard. Achieving each goal requires multiple actions of different types, including moving and acting on objects. This allows us to experiment with a simple variation of our model to generate intermediate goals.

We compare our approach to multiple recent methods. Experiments on the LANI navigation task indicate that decomposing goal prediction and action generation significantly improves instruction execution performance. While we observe similar trends on the CHAI instructions, results are overall weaker, illustrating the complexity of the task. We also observe that inherent ambiguities in instruction following make exact goal identification difficult, as demonstrated by imperfect human performance. However, the gap to human-level performance still remains large across both tasks. Our code and data are available at `https://github.com/clic-lab/ciff`.

6.2 Technical Overview

Task Let $\mathcal{X}$ be the set of all *instructions*, $\mathcal{S}$ the set of all *world states*, and $\mathcal{A}$ the set of all *actions*. An instruction $\bar{x} \in \mathcal{X}$ is a sequence $\langle x_1, \ldots, x_n \rangle$, where each x_i is a token. The agent executes instructions by generating a sequence of actions, and indicates execution completion with the special action `STOP`.

The sets of actions $\mathcal{A}$ and states $\mathcal{S}$ are domain specific. In the navigation domain LANI, the actions include moving the agent and changing its orientation. The state information includes the position and orientation of the agent and the different landmarks. The agent actions in the CHALET house environment include moving and changing the agent orientation, as well as an object interaction action. The state encodes the position and orientation of the agent and all objects in the house. For interactive objects, the state also includes their status, for example if a drawer is open or closed. In both domains, the actions are discrete. The domains are described in Section 6.5.

Model The agent does not observe the world state directly, but instead observes its pose and an RGB image of the environment from its point of view. We define these observations as the agent context $\tilde{s}$. An agent model is a function from an agent context $\tilde{s}$ to an action $a \in \mathcal{A}$. We model goal prediction as predicting a probability distribution over the agent visual observations, representing the likelihood of locations or objects in the environment being target positions or objects to be acted on. Our model is described in Section 6.3.

Learning We assume access to training data with N examples $\{(\bar{x}^{(i)}, s_1^{(i)}, s_g^{(i)})\}_{i=1}^{N}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $s_g^{(i)}$ is the goal state. We decompose learning; training goal prediction using supervised learning, and action generation using oracle goals with policy gradient in a contextual bandit setting. We assume an instrumented environment with access to the world state, which is used to compute rewards during training only. Learning is described in Section 6.4.

Evaluation We evaluate task performance on a test set $\{(\bar{x}^{(i)}, s_1^{(i)}, s_g^{(i)})\}_{i=1}^{M}$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $s_g^{(i)}$ is the goal state. We evaluate task completion accuracy and the distance of the agent's final state to $s_g^{(i)}$.

6.3 Model

We model the agent policy as a neural network. The agent observes the world state s_t at time t as an RGB image I_t. The agent context $\tilde{s}_t$, the information available to the agent to select the next action a_t, is a tuple $(\bar{x}, I_P, \langle (I_1, p_1), \ldots, (I_t, p_t) \rangle)$,

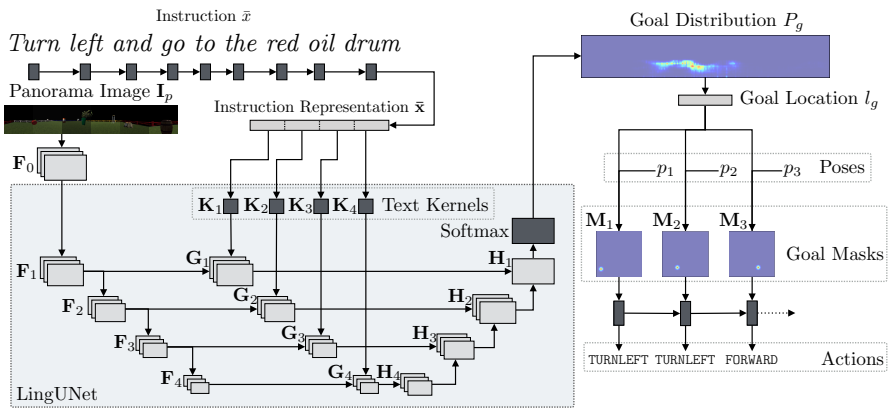


Figure 6.2: An illustration for our architecture (Section 6.3) for the instruction *turn left and go to the red oil drum* with a LINGUNETdepth of $m = 4$. The instruction $\bar{x}$ is mapped to $\bar{\mathbf{x}}$ with an RNN, and the initial panorama observation I_P to $\mathbf{F}_0$ with a CNN. LINGUNETgenerates $\mathbf{H}_1$, a visual representation of the goal. First, a sequence of convolutions maps the image features $\mathbf{F}_0$ to feature maps $\mathbf{F}_1,\ldots,\mathbf{F}_4$. The text representation $\bar{\mathbf{x}}$ is used to generate the kernels $\mathbf{K}_1,\ldots,\mathbf{K}_4$, which are convolved to generate the text-conditioned feature maps $\mathbf{G}_1,\ldots,\mathbf{G}_4$. These feature maps are de-convolved to $\mathbf{H}_1,\ldots,\mathbf{H}_4$. The goal probability distribution P_g is computed from $\mathbf{H}_1$. The goal location is the inferred from the max of P_g. Given l_g and p_t, the pose at step t, the goal mask $\mathbf{M}_t$ is computed and passed into an RNN that outputs the action to execute.

where $\bar{x}$ is the natural language instructions, I_P is a panoramic view of the environment from the starting position at time $t = 1$, and $\langle (I_1, p_1), \ldots, (I_t, p_t) \rangle$ is the sequence of observations I_t and poses p_t up to time t. The panorama I_P is generated through deterministic exploration by rotating 360° to observe the environment at the beginning of the execution.[1]

The model includes two main components: goal prediction and action generation. The agent uses the panorama I_P to predict the goal location l_g. At each time step t, a projection of the goal location into the agent's current view $\mathbf{M}_t$ is given as input to an RNN to generate actions. The probability of an action a_t at time t decomposes to:

[1]The panorama is a concatenation of deterministic observations along the width dimension. For simplicity, we do not include these deterministic steps in the execution.

$$P(a_t \mid \tilde{s}_t) = \sum_{l_g} \Big(P(l_g \mid \bar{x}, I_P) P(a_t \mid l_g, (I_1, p_1), \ldots, (I_t, p_t)) \Big) \ ,$$

where the first term puts the complete distribution mass on a single location (i.e., a delta function). Figure 6.2 illustrates the model.

Goal Prediction To predict the goal location, we generate a probability distribution P_g over a feature map $\mathbf{F}_0$ generated using convolutions from the initial panorama observation I_P. Each element in the probability distribution P_g corresponds to an area in I_P. Given the instruction $\bar{x}$ and panorama I_P, we first generate their representations. From the panorama I_P, we generate a feature map $\mathbf{F}_0 = [\text{CNN}_0(I_P); \mathbf{F}^p]$,

where CNN_0 is a two-layer convolutional neural network (CNN) [LeCun et al., 1998] with rectified linear units (ReLu) [Nair and Hinton, 2010] and $\mathbf{F}^p$ are positional embeddings.[2] The concatenation is along the channel dimension. The instruction $\bar{x} = \langle x_1, \cdots x_n \rangle$ is mapped to a sequence of hidden states $\mathbf{l}_i = \text{LSTM}_x(\psi_x(x_i), \mathbf{l}_{i-1})$, $i = 1, \ldots, n$ using a learned embedding function ψ_x and a long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] RNN LSTM_x. The instruction representation is $\bar{\mathbf{x}} = \mathbf{l}_n$.

We generate the probability distribution P_g over pixels in $\mathbf{F}_0$ using LINGUNET. The architecture of LINGUNETis inspired by the UNETimage generation method [Ronneberger et al., 2015], except that the reconstruction phase is conditioned on the natural language instruction. LINGUNETfirst applies m convolutional layers to generate a sequence of feature maps $\mathbf{F}_j = \text{CNN}_j(\mathbf{F}_{j-1})$,

[2]We generate $\mathbf{F}^p$ by creating a channel for each deterministic observation used to create the panorama, and setting all the pixels corresponding to that observation location in the panorama to 1 and all others to 0. The number of observations depends on the agent's camera angle.

$j = 1 \ldots m$, where each CNN_j is a convolutional layer with leaky ReLU non-linearities [Maas et al., 2013] and instance normalization [Ulyanov et al., 2016]. The instruction representation $\bar{\mathbf{x}}$ is split evenly into m vectors $\{\bar{\mathbf{x}}_j\}_{j=1}^m$, each is used to create a 1×1 kernel $\mathbf{K}_j = \text{AFFINE}_j(\bar{\mathbf{x}}_j)$, where each AFFINE_j is an affine transformation followed by normalizing and reshaping. For each $\mathbf{F}_j$, we apply a 2D 1×1 convolution using the text kernel $\mathbf{K}_j$ to generate a text-conditioned feature map $\mathbf{G}_j = \text{CONVOLVE}(\mathbf{K}_j, \mathbf{F}_j)$, where CONVOLVE convolves the kernel over the feature map. We then perform m deconvolutions to generate a sequence of feature maps $\mathbf{H}_m, \ldots, \mathbf{H}_1$:

$$\begin{aligned} \mathbf{H}_m &= \text{DECONV}_m(\text{DROPOUT}(\mathbf{G}_m)) \\ \mathbf{H}_j &= \text{DECONV}_j([\mathbf{H}_{j+1}; \mathbf{G}_j]) \ . \end{aligned}$$

DROPOUT is dropout regularization [Srivastava et al., 2014] and each DECONV_j is a deconvolution operation followed a leaky ReLU non-linearity and instance norm.[3] Finally, we generate P_g by applying a softmax to $\mathbf{H}_1$ and an additional learned scalar bias term b_g to represent events where the goal is out of sight. For example, when the agent already stands in the goal position and therefore the panorama does not show it.

We use P_g to predict the goal position in the environment. We first select the goal pixel in $\mathbf{F}_0$ as the pixel corresponding to the highest probability element in P_g. We then identify the corresponding 3D location l_g in the environment using backward camera projection, which is computed given the camera parameters and p_1, the agent pose at the beginning of the execution.

[3] DECONV_1 does deconvolution only.

Action Generation Given the predicted goal l_g, we generate actions using an RNN. At each time step t, given p_t, we generate the goal mask $\mathbf{M}_t$, which has the same shape as the observed image I_t. The goal mask $\mathbf{M}_t$ has a value of 1 for each element that corresponds to the goal location l_g in I_t. We do not distinguish between visible or occluded locations. All other elements are set to 0. We also maintain an out-of-sight flag o_t that is set to 1 if (a) l_g is not within the agent's view; or (b) the max scoring element in P_g corresponds to b_g, the term for events when the goal is not visible in I_P. Otherwise, o_t is set to 0. We compute an action generation hidden state y_t with an RNN:

$$y_t = \text{LSTM}_A\left(\text{AFFINE}_A([\text{FLAT}(\mathbf{M}_t); o_t]), y_{t-1}\right) \quad ,$$

where FLAT flattens $\mathbf{M}_t$ into a vector, AFFINE_A is a learned affine transformation with ReLU, and LSTM_A is an LSTM RNN. The previous hidden state y_{t-1} was computed when generating the previous action, and the RNN is extended gradually during execution. Finally, we compute a probability distribution over actions:

$$P(a_t \mid l_g, (I_1, p_1), \ldots, (I_t, p_t)) \quad = \quad \text{SOFTMAX}(\text{AFFINE}_p([y_t; \psi_T(t)])) \quad ,$$

where ψ_T is a learned embedding lookup table for the current time [Chaplot et al., 2018] and AFFINE_p is a learned affine transformation.

Model Parameters The model parameters θ include the parameters of the convolutions CNN_0 and the components of LINGUNET: CNN_j, AFFINE_j, and DECONV_j for $j = 1, \ldots, m$. In addition we learn two affine transformations AFFINE_A and AFFINE_p, two RNNs LSTM_x and LSTM_A, two embedding func-

tions ψ_x and ψ_T, and the goal distribution bias term b_g. In our experiments (Section 6.6), all parameters are learned without external resources.

6.4 Learning

Our modeling decomposition enables us to choose different learning algorithms for the two parts. While reinforcement learning is commonly deployed for tasks that benefit from exploration [Peters and Schaal, 2008, Mnih et al., 2013], these methods require many samples due to their high sample complexity. However, when learning with natural language, only a relatively small number of samples is realistically available. This problem was addressed in prior work by learning in a contextual bandit setting [Misra et al., 2017] or mixing reinforcement and supervised learning [Xiong et al., 2018]. Our decomposition uniquely offers to tease apart the language understanding problem and address it with supervised learning, which generally has lower sample complexity. For action generation though, where exploration can be autonomous, we use policy gradient in a contextual bandit setting [Misra et al., 2017].

We assume access to training data with N examples $\{(\bar{x}^{(i)}, s_1^{(i)}, s_g^{(i)})\}_{i=1}^N$, where $\bar{x}^{(i)}$ is an instruction, $s_1^{(i)}$ is a start state, and $s_g^{(i)}$ is the goal state. We train the goal prediction component by minimizing the cross-entropy of the predicted distribution with the gold-standard goal distribution. The gold-standard goal distribution is a deterministic distribution with probability one at the pixel corresponding to the goal location if the goal is in the field of view, or probability one at the extra out-of-sight position otherwise. The gold location is the agent's location in $s_g^{(i)}$. We update the model parameters using Adam [Kingma and Ba,

2014].

We train action generation by maximizing the expected immediate reward the agent observes while exploring the environment. The objective for a single example i and time stamp t is:

$$J = \sum_{a \in \mathcal{A}} \pi(a \mid \tilde{s}_t) R^{(i)}(s_t, a) + \lambda H(\pi(. \mid \tilde{s}_t)) \ ,$$

where $R^{(i)} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is an example-specific reward function, $H(\cdot)$ is an entropy regularization term, and λ is the regularization coefficient. The reward function $R^{(i)}$ details are described in details in Appendix C.2. Roughly speaking, the reward function includes two additive components: a problem reward and a shaping term [Ng et al., 1999]. The problem reward provides a positive reward for successful task completion, and a negative reward for incorrect completion or collision. The shaping term is positive when the agent gets closer to the goal position, and negative if it is moving away. The gradient of the objective is:

$$\nabla J = \sum_{a \in \mathcal{A}} \pi(a \mid \tilde{s}_t) \nabla \log \pi(a \mid \tilde{s}_t) R(s_t, a) + \lambda \nabla H(\pi(. \mid \tilde{s}_t) \ .$$

We approximate the gradient by sampling an action using the policy [Williams, 1992], and use the gold goal location computed from $s_g^{(i)}$. We perform several parallel rollouts to compute gradients and update the parameters using Hogwild! [Recht et al., 2011] and Adam learning rates.

Dataset Statistic	LANI	CHAI
Number paragraphs	6,000	1,596
Mean instructions per paragraph	4.7	7.70
Mean actions per instruction	24.6	54.5
Mean tokens per instruction	12.1	8.4
Vocabulary size	2,292	1,018

Table 6.1: Summary statistics of the two corpora.

6.5 Tasks and Data

6.5.1 LANI

The goal of LANI is to evaluate how well an agent can follow navigation instructions. The agent task is to follow a sequence of instructions that specify a path in an environment with multiple landmarks. Figure 6.1 (left) shows an example instruction.

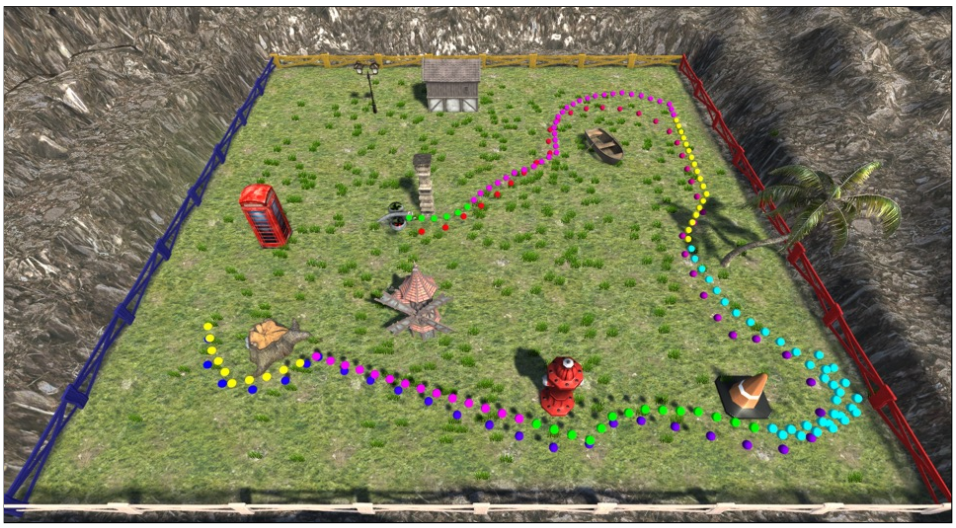
The environment is a fenced, square, grass field. Each instance of the environment contains between 6–13 randomly placed landmarks, sampled from 63 unique landmarks. The agent can take four types of discrete actions: `FORWARD`, `TURNRIGHT`, `TURNLEFT`, and `STOP`. The field is of size 50×50, the distance of the `FORWARD` action is 1.5, and the turn angle is 15°. The environment simulator is implemented in Unity3D. At each time step, the agent performs an action, observes a first person view of the environment as an RGB image, and receives a scalar reward. The simulator provides a socket API to control the agent and the environment.

Agent performance is evaluated using two metrics: task completion accuracy, and stop distance error. A task is completed correctly if the agent stops within an aerial distance of 5 from the goal.

We collect a corpus of navigation instructions using crowdsourcing. We randomly generate environments, and generate one reference path for each environment. To elicit linguistically interesting instructions, reference paths are generated to pass near landmarks. We use Amazon Mechanical Turk, and split the annotation process to two tasks. First, given an environment and a reference path, a worker writes an instruction paragraph for following the path. The second task requires another worker to control the agent to perform the instructions and simultaneously mark at each point what part of the instruction was executed. The recording of the second worker creates the final data of segmented instructions and demonstrations. The generated reference path is displayed in both tasks. The second worker could also mark the paragraph as invalid. Both tasks are done from an overhead view of the environment, but workers are instructed to provide instructions for a robot that observes the environment from a first person view. Figure 6.3 shows a reference path and the written instruction. This data can be used for evaluating both executing sequences of instructions and single instructions in isolation.

Table 6.1 shows the corpus statistics.[4] Each paragraph corresponds to a single unique instance of the environment. The paragraphs are split into train, test, and development, with a 70% / 15% / 15% split. Finally, we sample 200 single development instructions for qualitative analysis of the language challenge the corpus presents (Table 6.2).

[4] Appendix C.1 provides statistics for related datasets.



[*Go around the pillar on the right hand side*] [*and head towards the boat, circling around it clockwise.*] [*When you are facing the tree, walk towards it, and the pass on the right hand side,*] [*and the left hand side of the cone. Circle around the cone,*] [*and then walk past the hydrant on your right,*] [*and the the tree stump.*] [*Circle around the stump and then stop right behind it.*]

Figure 6.3: Segmented instructions in the LANI domain. The original reference path is marked in red (start) and blue (end). The agent, using a drone icon, is placed at the beginning of the path. The follower path is coded in colors to align to the segmented instruction paragraph.

6.5.2 CHAI

The CHAI corpus combines both navigation and simple manipulation in a complex, simulated household environment. We use the CHALET simulator [Yan et al., 2018], a 3D house simulator that provides multiple houses, each with multiple rooms. The environment supports moving between rooms, picking and placing objects, and opening and closing cabinets and similar containers. Objects can be moved between rooms and in and out of containers. The agent observes the world in first-person view, and can take five actions: `FORWARD`, `TURNLEFT`, `TURNRIGHT`, `STOP`, and `INTERACT`. The `INTERACT` action acts on objects. It takes as argument a 2D position in the agent's view. Agent performance is evaluated with two metrics: (a) stop distance, which measures the distance of the agent's final state to the final annotated position; and (b) manipulation ac-

Category	Count		Example
	LANI	CHAI	
Spatial relations between locations	123	52	LANI: *go to the* ***right side of the*** *rock* CHAI: *pick up the cup* ***next to the*** *bathtub and place . . .*
Conjunctions of two or more locations	36	5	LANI: *. . . between* ***the mushroom and the yellow cone*** CHAI: *. . . on the table next to* ***the juice and milk****.*
Temporal coordination of sub-goals	65	68	LANI: *at the mushroom* ***turn right and move forward towards the statue*** CHAI: ***go back to*** *the kitchen* ***and put the glass in the sink.***
Constraints on the shape of trajectory	94	0	LANI: *go past the house* ***by the right side of the apple***
Co-reference	32	18	LANI: *turn around* ***it*** *and move in front of fern plant* CHAI: *turn left, towards the kitchen door and move through* ***it****.*
Comparatives	2	0	LANI: *. . . the small stone* ***closest to the blue and white fences*** *stop*

Table 6.2: Qualitative analysis of the LANI and CHAI corpora. We sample 200 single development instructions from each corpora. For each category, we count how many examples of the 200 contained it and show an example.

Scenario
You have several hours before guests begin to arrive for a dinner party. You are preparing a wide variety of meat dishes, and need to put them in the sink. In addition, you want to remove things in the kitchen, and bathroom which you don't want your guests seeing, like the soaps in the bathroom, and the dish cleaning items. You can put these in the cupboards. Finally, put the dirty dishes around the house in the dishwasher and close it.
Written Instructions
[*In the kitchen, open the cupboard above the sink.***]** **[***Put the cereal, the sponge, and the dishwashing soap into the cupboard above the sink.***]** **[***Close the cupboard.***]** **[***Pick up the meats and put them into the sink.***]** **[***Open the dishwasher, grab the dirty dishes on the counter, and put the dishes into the dishwasher.***]**

Figure 6.4: Scenario and segmented instruction from the CHAI corpus.

curacy, which compares the set of manipulation actions to a reference set. When measuring distance, to consider the house plan, we compute the minimal aerial distance for each room that must be visited. Yan et al. [2018] provides the full details of the simulator and evaluation. We use five different houses, each with up to six rooms. Each room contains on average 30 objects. A typical room is of size 6×6. We set the distance of `FORWARD` to 0.1, the turn angle to 90°, and divide the agent's view to a 32×32 grid for the `INTERACT` action.

We collected a corpus of navigation and manipulation instructions using Amazon Mechanical Turk. We created 36 common household scenarios to provide a familiar context to the task.[5] We use two crowdsourcing tasks. First, we provide workers with a scenario and ask them to write instructions. The workers are encouraged to explore the environment and interact with it. We then segment the instructions to sentences automatically. In the second task, workers are presented with the segmented sentences in order and asked to execute them. After finishing a sentence, the workers request the next sentence. The workers do not see the original scenario. Figure 6.4 shows a scenario and the written segmented paragraph. Similar to LANI, CHAI data can be used for studying complete paragraphs and single instructions.

Table 6.1 shows the corpus statistics.[6] The paragraphs are split into train, test, and development, with a 70% / 15% / 15% split. Table 6.2 shows qualitative analysis of a sample of 200 instructions.

6.6 Experimental Setup

Method Adaptations for CHAI We apply two modifications to our model to support intermediate goal for the CHAI instructions. First, we train an additional RNN to predict the sequence of intermediate goals given the instruction only. There are two types of goals: `NAVIGATION`, for action sequences requiring movement only and ending with the `STOP` action; and `INTERACTION`, for se-

[5] We observed that asking workers to simply write instructions without providing a scenario leads to combinations of repetitive instructions unlikely to occur in reality.

[6] The number of actions per instruction is given in the more fine-grained action space used during collection. To make the required number of actions smaller, we use the more coarse action space specified.

quence of movement actions that end with an INTERACT action. For example, for the instruction *pick up the red book and go to the kitchen*, the sequence of goals will be ⟨INTERACTION, NAVIGATION, NAVIGATION⟩. This indicates the agent must first move to the object to pick it up via interaction, move to the kitchen door, and finally move within the kitchen. The process of executing an instruction starts with predicting the sequence of goal types. We call our model (Section 6.3) separately for each goal type. The execution concludes when the final goal is completed. For learning, we create a separate example for each intermediate goal and train the additional RNN separately. The second modification is replacing the backward camera projection for inferring the goal location with ray casting to identify INTERACTION goals, which are often objects that are not located on the ground.

Baselines We compare our approach against the following baselines: (a) STOP: Agent stops immediately; (b) RANDOMWALK: Agent samples actions uniformly until it exhausts the horizon or stops; (c) MOSTFREQUENT: Agent takes the most frequent action in the data, FORWARD for both datasets, until it exhausts the horizon; (d) MISRA17: the approach of Misra et al. [2017]; and (e) CHAPLOT18: the approach of Chaplot et al. [2018]. We also evaluate goal prediction and compare to the method of Janner et al. [2018] and a CENTER baseline, which always predict the center pixel. Appendix C.3 provides baseline details.

Evaluation Metrics We evaluate using the metrics described in Section 6.5: stop distance (SD) and task completion (TC) for LANI, and stop distance (SD) and manipulation accuracy (MA) for CHAI. To evaluate the goal prediction, we report the real distance of the predicted goal from the annotated goal and the

Method	LANI		CHAI	
	SD	TC	SD	MA
STOP	15.37	8.20	2.99	37.53
RANDOMWALK	14.80	9.66	2.99	28.96
MOSTFREQUENT	19.31	2.94	3.80	37.53
MISRA17	10.54	22.9	2.99	32.25
CHAPLOT18	9.05	31.0	2.99	37.53
Our Approach (OA)	**8.65**	**35.72**	**2.75**	37.53
OA w/o RNN	9.21	31.30	3.75	37.43
OA w/o Language	10.65	23.02	3.22	37.53
OA w/joint	11.54	21.76	2.99	36.90
OA w/oracle goals	2.13	94.60	2.19	41.07

Table 6.3: Performance on the development data.

percentage of correct predictions. We consider a goal correct if it is within a distance of 5.0 for LANI and 1.0 for CHAI. We also report human evaluation for LANI by asking raters if the generated path follows the instruction on a Likert-type scale of 1–5. Raters were shown the generated path, the reference path, and the instruction.

Parameters We use a horizon of 40 for both domains. During training, we allow additional 5 steps to encourage learning even after errors. When using intermediate goals in CHAI, the horizon is used for each intermediate goal separately. All other parameters and detailed in Appendix C.4.

6.7 Results

Tables 6.3 and 6.4 show development and test results. Both sets of experiments demonstrate similar trends. The low performance of STOP, RANDOMWALK, and MOSTFREQUENT demonstrates the challenges of both tasks, and shows the tasks are robust to simple biases. On LANI, our approach outperforms CHAP-

Method	LANI		CHAI	
	SD	TC	SD	MA
STOP	15.18	8.29	3.59	39.77
RANDOMWALK	14.63	9.76	3.59	33.29
MOSTFREQUENT	19.14	3.15	4.36	39.77
MISRA17	10.23	23.2	3.59	36.84
CHAPLOT18	8.78	31.9	3.59	39.76
Our Approach	**8.43**	**36.9**	**3.34**	**39.97**

Table 6.4: Performance on the held-out test dataset.

Method	LANI		CHAI	
	Dist	Acc	Dist	Acc
CENTER	12.0	19.51	3.41	19.0
Janner et al. [2018]	9.61	30.26	2.81	28.3
Our Approach	8.67	35.83	2.12	40.3

Table 6.5: Development goal prediction performance. We measure distance (Dist) and accuracy (Acc).

LOT18, improving task completion (TC) accuracy by 5%, and both methods outperform MISRA17. On CHAI, CHAPLOT18 and MISRA17 both fail to learn, while our approach shows an improvement on stop distance (SD). However, all models perform poorly on CHAI, especially on manipulation (MA).

To isolate navigation performance on CHAI, we limit our train and test data to instructions that include navigation actions only. The STOP baseline on these instructions gives a stop distance (SD) of 3.91, higher than the average for the entire data as these instructions require more movement. Our approach gives a stop distance (SD) of 3.24, a 17% reduction of error, significantly better than the 8% reduction of error over the entire corpus.

We also measure human performance on a sample of 100 development examples for both tasks. On LANI, we observe a stop distance error (SD) of 5.2 and successful task completion (TC) 63% of the time. On CHAI, the human dis-

Category	Present	Absent	p-value
Spatial relations	8.75	10.09	.262
Location conjunction	10.19	9.05	.327
Temporal coordination	11.38	8.24	.015
Trajectory constraints	9.56	8.99	.607
Co-reference	12.88	8.59	.016
Comparatives	10.22	9.25	.906

Table 6.6: Mean goal prediction error for LANI instructions with and without the analysis categories we used in Table 6.2. The p-values are from two-sided t-tests comparing the means in each row.

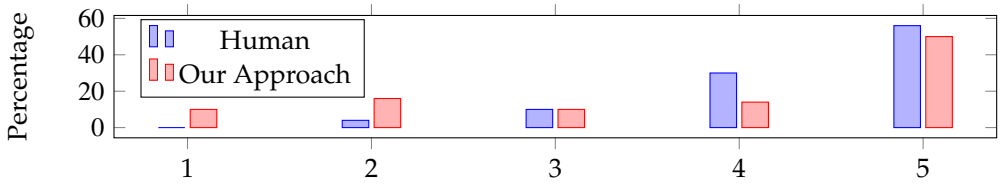


Figure 6.5: Likert rating histogram for expert human follower and our approach for LANI.

tance error (SD) is 1.34 and the manipulation accuracy is 100%. The imperfect performance demonstrates the inherent ambiguity of the tasks. The gap to human performance is still large though, demonstrating that both tasks are largely open problems.

The imperfect human performance raises questions about automated evaluation. In general, we observe that often measuring execution quality with rigid goals is insufficient. We conduct a human evaluation with 50 development examples from LANI rating human performance and our approach. Figure 6.5 shows a histogram of the ratings. The mean rating for human followers is 4.38, while our approach's is 3.78; we observe a similar trend to before with this metric. Using judgements on our approach, we correlate the human metric with the SD measure. We observe a Pearson correlation -0.65 (p=5e-7), indicating that our automated metric correlates well with human judgment.[7] This initial study

[7]We did not observe this kind of clear anti-correlation comparing the two results for human

suggests that our automated evaluation is appropriate for this task.

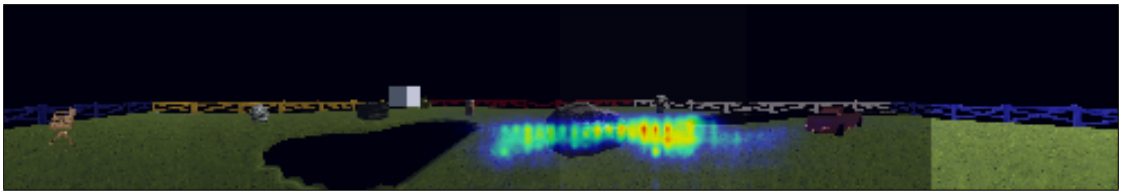
Our ablations (Table 6.3) demonstrate the importance of each of the components of the model. We ablate the action generation RNN (w/o RNN), completely remove the language input (w/o Language), and train the model jointly (w/joint Learning).[8] On CHAI especially, ablations results in models that display ineffective behavior. Of the ablations, we observe the largest benefit from decomposing the learning and using supervised learning for the language problem.

We also evaluate our approach with access to oracle goals (Table 6.3). We observe this improves navigation performance significantly on both tasks. However, the model completely fails to learn a reasonable manipulation behavior for CHAI. This illustrates the planning complexity of this domain. A large part of the improvement in measured navigation behavior is likely due to eliminating much of the ambiguity the automated metric often fails to capture.

Finally, on goal prediction (Table 6.5), our approach outperforms the method of Janner et al. [2018]. Figure 6.6 and Appendix Figure C.1 show example goal predictions. In Table 6.6, we break down LANI goal prediction results for the analysis categories we used in Table 6.2 using the same sample of the data. Appendix C.5 includes a similar table for CHAI. We observe that our approach finds instructions with temporal coordination or co-reference challenging. Co-reference is an expected limitation; with single instructions, the model can not resolve references to previous instructions.

performance (Pearson correlation of 0.09 and p=0.52). The limited variance in human performance makes correlation harder to test.

[8] Appendix C.3 provides the details of joint learning.



curve around big rock keeping it to your left .

walk over to the cabinets and open the cabinet doors up

Figure 6.6: Goal prediction probability maps P_g overlaid on the corresponding observed panoramas I_P. The top example shows a result on LANI, the bottom on CHAI.

6.8 Discussion

We propose a model for instruction following with explicit separation of goal prediction and action generation. Our representation of goal prediction is easily interpretable, while not requiring the design of logical ontologies and symbolic representations. A potential limitation of our approach is cascading errors. Action generation relies completely on the predicted goal and is not exposed to the language otherwise. This also suggests a second related limitation: the model is unlikely to successfully reason about instructions that include constraints on the execution itself. While the model may reach the final goal correctly, it is unlikely to account for the intermediate trajectory constraints. As we show (Table 6.2), such instructions are common in our data. These two limitations may be addressed by allowing action generation access to the instruction. Achieving this while retaining an interpretable goal representation that clearly determines the execution is an important direction for future work. Another important open

question concerns automated evaluation, which remains especially challenging when instructions do not only specify goals, but also constraints on how to achieve them. Our resources provide the platform and data to conduct this research.

CHAPTER 7

CORNELL INSTRUCTION FOLLOWING FRAMEWORK (CIFF)

7.1 Introduction

We introduced several tasks in this thesis including manipulating blocks on a map, navigating in 3D environments and performing manipulation in a 3D house. Many other environments have been proposed recently for instruction following [Hermann et al., 2017, Chaplot et al., 2018, Anderson et al., 2018, Blukis et al., 2018b, Das et al., 2018, Chen et al., 2019].

In order to trust the performance of an agent, it is important to evaluate it on a multitude of tasks. This reduces the probability of overfitting to a specific task by exploiting task-specific biases. However, experimenting with multiple tasks is often a time consuming effort requiring downloading the datasets separately, parsing them to a common format and tuning the model and learning algorithms to handle the nuances of the specific task. To reduce this engineering effort we introduce an integrated learning framework for instruction following called the *Cornell Instruction Following Framework* (CIFF). CIFF provides an abstract interface for datasets, environments, learning algorithms and models to communicate with each other. Existing implementation of models and learning algorithms can be used for a new task as long as the new task implements the CIFF interface.

7.2 Features of CIFF

CIFF provides an abstract interface and comes with four different instruction following tasks. CIFF also contains implementation of various models and learning algorithm. We now describe the different features of the CIFF.

- **CIFF Interface:** The central concept in CIFF is an interface that allows models, learning algorithm, simulators and dataset to interact with each other in a task agnostic way. There is an interface for representing a datapoint, a server for interacting with the simulator, a validation class for verifying the setting, and a data loader for reading the dataset. Also, there are a few interfaces for models and learning algorithms.
- **Models:** CIFF contains implementation of various models and baselines including the ones introduced in Chapter 5 and Chapter 6. A model takes an agent context as input and generates either a probability distribution over the action space or value function estimates. CIFF also provides implementation of several modules which act as building blocks for larger models and allow code reusability. For example, LSTMs [Hochreiter and Schmidhuber, 1997] and ResNets [He et al., 2016] are implemented as modules and are used by several different models.
- **Distributed Learning Framework:** CIFF provides a distributed learning framework for efficient learning. The agent can interact with multiple simulators in parallel allowing it to collect data efficiently and reduce training and testing time. CIFF allows user to specify the number of processes that needs to be launched. It then launches that many copies of the simulator and connects the agent to each one of the simulators using socket connection. If the agent contains a model then a copy of the model is transferred

to each one of these processes while the original master process retains the master copy that is used for evaluation. The agent interacts with each process separately and computes gradients with respect to the local model parameters. These gradients are then copied to the master process which applies Hogwild! [Recht et al., 2011] updates to the master copy. Hogwild! updates do not put a lock on the model parameters and reduces training time.

Implementation CIFF source code is implemented using Python3 programming language. The neural network models are implemented using PyTorch library [pyt] which provides support for dynamic networks, GPUs and multiprocessing.

Using CIFF CIFF implementation is made available under GNU General Public License v3.0 license at this link: `https://www.github.com/clic-lab/ciff`.

CHAPTER 8

CONCLUSION

In this thesis, we introduced new methods for the instruction following task. In Chapter 4 we used a graphical model to map high-level instructions to actions in a 3D house environment. While this approach can outperform baselines and even ground unseen verbs to their meaning, it required expensive feature engineering and access to the symbolic representation of the environment. To overcome this limitation, in Chapter 5 we introduced a single model method for mapping instruction and raw RGB images to actions for manipulating blocks on a map. Our approach does not require any expensive feature engineering or access to symbolic representation of the environment. We used a neural network to model the agent's policy and introduced a policy gradient algorithm with contextual bandit approximation to train the agent. Our approach is able to outperform supervised learning and common reinforcement learning baselines. However, the model we proposed in Chapter 5 is hard to reason about. This lack of interpretability also raises concerns about the safety of these systems. In Chapter 6 we introduced a neural network model that can provide evidence of understanding even prior to taking actions. Our model separates the problem of predicting the goal given the natural language instruction and visual observations, and generating actions to achieve the goal. The goal prediction can be used for interpreting the model and could be used to enhance the safety of these systems by preventing the agent to generate actions if the predicted goal is unsafe.

Many different environments and corpuses have been recently introduced for the instruction following problem [Hermann et al., 2017, Chaplot et al.,

2018, Das et al., 2018, Gordon et al., 2018, Chen et al., 2019]. In order to avoid overfitting to biases of a specific task, it is important to evaluate on multiple tasks. In Chapter 7 we introduced the Cornell Instruction Following Framework (CIFF) – an integrated learning framework for instruction following. CIFF comes with implementation of several existing models, learning algorithms, and tasks. Further, it provides an abstract interface that enables the existing models and learning algorithms to be applied to any new tasks that implements this interface. We make CIFF publicly available for experiments at `https://github.com/clic-lab/ciff`.

8.1 Future Directions

The instruction following task presented in this thesis can be extended along several directions. We briefly describe four future work directions below.

Task Based Dialogue In the instruction following task we considered, the agent has to follow natural language instructions but cannot ask for clarification. However, in real life humans often ask for clarification when the instruction is unclear or when it is misaligned with the environment. For example, consider the instruction *"get me some milk from the fridge"*. The robot may realize on opening the fridge that there is no milk left in the kitchen. In that case, the robot can communicate this to the user and ask for further instructions.

This setting raises challenging problems of realizing when to ask a question, what question to ask, how to utilize the user feedback, and how to learn from long sequence of interactions. Task based dialogue have been thoroughly studied in other settings, for example, booking flight tickets [Peng

et al., 2017], restaurant reservations [Su et al., 2016], and boostrapping semantic parsers [Artzi and Zettlemoyer, 2011]. However, dialogue agents in visually grounded settings haven't been thoroughly studied.

Evaluating Instruction Following Agents Evaluating instruction following agents can be challenging. In Chapter 4 we evaluate the agent using the Levenshtein edit-distance between the agent's trajectory and the gold trajectory. In Chapter 5 and Chapter 6 we use stop-distance metric that computes the distance between the final state of the world after agent's execution and the gold final state. In general, both of these evaluation metrics can be wrong. Consider the following two instructions for navigation: *"go towards the barrel"* and *"go towards the barrel while passing by the tree"*. Consider the gold trajectories for the two instructions: for the first instruction the agent goes directly towards the barrel while never passing by the tree. For the second instruction, the agent passes by the tree and stops near the barrel. For the first instruction, both trajectories are correct while for the second instruction only the second trajectory is correct. For the first instruction, the edit-distance metric will penalize the second-trajectory even though it is correct. For the second instruction, the stop-distance metric will incentivize the first trajectory even though it is wrong. While one can use human evaluation, it can be impractical to scale. Designing an automated evaluation metric that correlate well with human judgment is an interesting direction for future work.

Designing Algorithms for Strategic Exploration Instruction following tasks can require complex planning with long sequence of actions and a large action space. Performing train time and test time exploration in this setting can

be, therefore, challenging. The recent work on instruction following, including the algorithms in Chapter 5 and Chapter 6, utilizes on-policy exploration. On-policy exploration uses the same policy for performing exploration and optimizing the reward. On-policy methods can easily fail to solve problems requiring *strategic exploration* [Kakade and Langford, 2002]. In the reinforcement learning literature, the problem of designing algorithms for strategic exploration is both central and widely studied. Several algorithms exist for tabular Markov Decision Processes (MDPs) that can perform strategic exploration with theoretical guarantees including the E^3 algorithm [Kearns and Singh, 2002], Delayed Q-Learning [Strehl et al., 2006], R_{max} [Brafman and Tennenholtz, 2002] and MBIE-EB [Strehl and Littman, 2008]. However, approaches for non-tabular setting lack theoretical guarantees or make specific assumptions about the problem [Du et al., 2019]. Designing theoretical provable algorithms for strategic exploration in complex real world tasks remains an open challenge.

Mapping Natural Language Feedback to Reward In this thesis we consider a version of the instruction following task where we have access to a validation function for evaluating the agent. However, in general such a validation function may not be available. For example, a voice assistant application such as Cortana, Siri, Alexa or Google Voice Assistant, do not receive a scalar reward at the end of their interaction from users. However, getting natural language feedback is practical. The user feedback can contain a 0-1 evaluation as in the instruction, *"you failed to do the task"*, or richer signal such as in the instruction, *"you picked the wrong object. You were supposed to bring me the cup next to it."*. In order to train the agents, we need to convert this feedback to a validation function. This validation function could output a scalar score or a set of features.

This could also potentially alleviate privacy concerns by removing the need to store raw human feedback in the cloud and instead storing just the anonymized validation function.

APPENDIX A

APPENDIX FOR CHAPTER 4

A.1 Parsing Text into Control Flow Graph.

We first decompose the text $\bar{x}$ into its control flow graph G using a simple set of rules:

- The parse tree of $\bar{x}$ is generated using the Stanford parser [Klein and Manning, 2003] and a frame node is created for each non-auxiliary verb node in the tree.
- Conditional nodes are discovered by looking for the keywords *until, if, after, when*. The associated subtree is then parsed deterministically using a set of a rules. For example, a rule parses *"for x minutes"* to `for(digit:x,unit:minutes)`. We found that all conditionals can be interpreted against the initial environment s_1, since our world is fully-observable, deterministic, and the user giving the command has full view of the world.
- To find objects, we look for anaphoric terminal nodes or nominals whose parent is not a nominal or which have a PP sibling. These are processed into object descriptions ω.
- Object descriptions ω are attached to the frame node, whose verb is nearest in the parse tree to the main noun of ω.
- Nodes corresponding to *{IN,TO,CC,",",}* are added as the relation between the corresponding argument objects.

- If there is a conjunction between two objects in a frame node and if these objects have the same relation to other objects, then we split the frame node into two sequential frame nodes around these objects. For example, a frame node corresponding to the text segment "*take the cup and bowl from table*" is split into two frame nodes corresponding to "*take the cup from table*" and "*take bowl from table*".
- A temporal edge is added between successive frame nodes in the same branch of a condition. A temporal edge is added between a conditional node and head of the true and false branches of the condition. The end of all branches in a sentence are joined to the starting node of the successive sentence.

A.2 Dataset: Samples and Challenges

As described in the main paper, we collected a dataset $D = (\bar{x}^{(n)}, s^{(n)}, a^{(n)}, \beta^{(n)})_{n=1}^{500}$.

Environment Complexity. Our environments are 3D scenarios consisting of complex objects such as fridge, microwave and television with many states. These objects can be in different spatial relations with respect to other objects. For example, *"bag of chips"* can be found behind the television. Figure A.1 shows some sample environments from our dataset. For example, an object of category television consists of 6 channels, volume level and power status. An object can have different values of states in different environment and different environment consists of different set of objects and their placement. For example, television might be powered on in one environment and closed in another, mi-

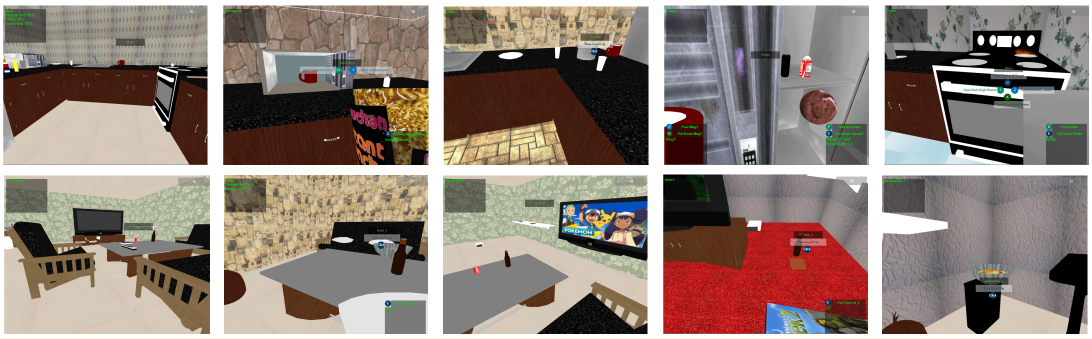


Figure A.1: Sample of 3D Environments that we consider. Environments consists of several objects, each object can have several states. Different environment have different set of objects with different configuration. There can be more than one objects of the same category.

crowave might have an object inside it or not in different environment, etc.

Moreover, there are often more than one object of the same category. For example, our environment typically have two books, five couches, four pillows etc. Objects of the same category can have different appearance. For example, a book can have the cover of a math book or a Guinness book of world record; resulting in complex object descriptions such as in *"throw the sticky stuff in the bowl"*. They can also have the same appearance making people use spatial relations or other means while describing them such as in *"get me the cup next to the microwave"*. This dataset is significantly more challenging compared to the 2D navigation dataset or GUI actions in windows dataset considered earlier.

Task Complexity. In this paper, we consider tasks with high level objective such as *clean the room, prepare the room for movie night* etc. compared to navigation or simple manipulations tasks involving picking and placing objects. This results in extremely free-form text such as shown below:

- *"Turn on xbox. Take Far Cry Game CD and put in xbox by pressig eject to open drive. Throw out beer, coke, and sketchy stuff in bowl. Take pillows from shelf and distribute among couches ."*
- *"Boil some water and make some coffee. Find a white bowl. Take ice cream out of the freezer. Put coffee into the white bowl, then put two scoops of ice cream over that . Finally, take the syrup on the counter and drizzle it over the ice cream."*
- *"If anything is disposable and used, put it in the trash bag. If it is not disposable and on the floor, put it on the table nearest any items associated with it. If it is not disposable and on the floor, put it on the table nearest any items associated with it. If it is not disposable and on the floor, put it on the table nearest any items associated with it. If a not disposable item contains only disposable objects, dump them into the trash bag, and treat the object like it was on the floor. Remove the trash bag from the scenario."*
- *"Make some coffee. Make some eggs on the stove and then put them on a plate and serve the eggs and coffee to me "*
- *"Take Book of Records and place on table with brown book. The TV is already turned off .Throw out open beer and coke. Chips are good."*
- *"Dump the coffee in the mug in the sink, put all perishable items in the refrigerator, put all the dishes, utensils, and pots in the sink."*
- *"Turn TV on with remote and find movie (Lincoln) on with remote and find movie (Lincoln) . Take bag of chips and place on table. Take pillow from shelf and place on a sofa. Throw away beer and soda, and place Book of Records on shelf with brown book."*
- *"Mix syrup and water to make a drink. You can get water by rotating the tab near sink. Use kettle to boil water and mix heated water with instantRamen."*

we refer the readers to the full dataset for more examples.

Noise in the dataset. Our dataset was collected from non-expert users including the action sequences. Therefore, our dataset had considerable noise as is also visible from the examples above. The noise included spelling and grammar errors in the text, text that is asking the robot to do things which it cannot do such as moving the chairs, noise in the action sequences and noise in aligning parts of action sequences and the text segments.

We use a set of rules to remove noise from the dataset, such as removing cyclic patterns in the action sequence. This often happened when users tried to give a demonstration to the robot such as keeping a mug inside the microwave, but made an error and hence repeated the actions. We want to emphasize here that, the average length of 21.5 actions for the action sequences in the dataset was derived after removing this noise.

Out of the 500 points that we collected, we further removed 31 points consisting of action sequences of length less than 2.

A.3 Examples of Planning and Simulation

We use a planner and a simulator which allows us to use post-conditions in defining our logical forms. In order to perform planning and simulation— we encode the domain knowledge in STRIPS planning format. This defines preconditions and effect of action on the environment. An example is given below:

(*:action* `release` *:parameters* (*o*)

:precondition (`grasping robot o`)

:effect (`not (grasping robot o)`)))

This STRIPS program defines the action `release` which takes an object o as the argument. The precondition of this action is that the robot must be grasping the object o and the effect is that robot releases the object o.

A.4 Mapping Object Descriptions

Given an object description ω and a set of physical objects $\{o_j\}_{j=1}^{m}$; we want to find the correlation $\rho(\omega, o_j) \in [0, 1]$ of how well does the description ω describes the object o_j. When the description is not a pronoun, we take the following approach. We initialize $\forall_j\ \rho(\omega, o_j) = 0$ and then try the following rules in the given order, stopping after the first match:

- *category matching*: if there exists a set of objects $\{o'_j\}$ containing part of the description in its name then we define $\forall_j \rho(\omega, o'_j) = 1$.
- *containment (metonymy)*: for every object o_j; if the main noun in ω matches the state-name of a state of o_j which has value $True$ then we define $\rho(\omega, o_j) = 1$.
- *wordnet similarity*: for every object o_j we find $\rho(\omega, o_j)$ using a modified Lesk algorithm based on WordNet. If a similarity score greater than 0.85 is found then we return.
- *domain specific references*: We use giza-pp algorithm to learn translation probabilities between text and corresponding action sequences, using the training data. This gives us a probability table T[*words,object-name*] of

words in text and object name in the sequence. We then initialize $\rho(\omega, o_j)$ by averaging the value of $T[w, o_j.name]$ for every word w in ω.

A.5 Manual Rules for Parsing Conditions

As explained in the paper, we parse conditional expressions into their meaning representations using a set of rules. This was possible and motivated both by the fact that the conditional expressions in our dataset are easy and because meaning representations of conditional expressions are not observed in the dataset (which only contains actions corresponding to frame nodes). We parse conditional expressions using the following deterministic rules.

string which is a noun or a pronoun → *object*

string representing a state-name → *statename*

string representing a spatial relation → *relation*

"minute"|"min"|"hour"|"sec"|"seconds" → time-unit

object statename → *state(object, statename)*

string1 relation string2 → *relation(string1, string2)*

digit time-unit → *time(digit, time-unit)*

for/when/after/until *state(object, statename)* →
for/when/after/until(state(object, statename))

for/after/until *time(digit,unit)* → *for/after/until(time(digit,unit))*

if *state(string1, string2)* → *if(state(string1, string2))*

Each word in the text can further be ignored, i.e., mapped to ϵ. These rules are simple enough to be parsed in a bottom up fashion starting with words. For

example, *"for 3 minutes"* is parsed as:

minutes→ time-unit:min

3 time-unit:min→ time(digit:3,time-unit:min)

for time(3,time-unit:min)→ for(time(digit:3,time-unit:min))

For *"if"* condition, which has a true and false branch; we evaluate the condition using the starting environment. In case of a parsing failure, we always return true.

A.6 Feature Equation

We use the following features $\phi(c_i, z_{i-1}, z_i, s_i)$ briefly described in the paper. The logical form is given by $z_i = ([v \Rightarrow (\lambda\vec{v}.S, \xi)], \xi_i)$. Here ξ_i, ξ are mappings of the variable $\vec{v}$ of the parametrized post-condition S. Let $\vec{v}$ have m variables and $\xi(v_j)$ represents the object in s_i, to which the variable v_j is mapped using ξ. Further the post-condition f_i is given by $f_i = (\lambda\vec{v}.S)(\xi_i)$:

- *Language and Logical Form*: There are two features of this type:

$$f_{le} = \frac{1}{m}\sum_{j=1}^{m} \max_{\omega} \rho(\omega, \xi_i(v_j))$$

$$f_{recall} = \frac{1}{|\omega \in c_i|}\sum_{\omega \in c_i} \max_{j} \rho(\omega, \xi_i(v_j))$$

  where ρ is the object description correlation score(see paper). For the f_{LE} feature, we also consider the previous clause c_{i-1} in the computation of $\max_{\omega} \rho(\omega, \xi_i(v_j))$.

- *Logical Form*: We prefer the post-conditions which have high environment priors and are therefore likely to occur again. Let post-condition $f_i = \wedge_l f_{il} = f_{i1} \wedge f_{i2} \cdots f_{ip}$ consists of p atomic-predicates (or their negations) given by f_{il}. Also let, $pm(\wedge_l f_{il})$ be the parametrized version of the post-condition $\wedge_l f_{il}$ created by replacing each unique object by a unique variable. Example, the post-condition $on(cup_2, bowl_3) \wedge state(cup_2, water)$ is parametrized to $on(v_1, v_2) \wedge state(v_1, water)$.

  We capture this property by $4t$ features where t denotes the maximum number of predicates that we consider simultaneously for creating the probability tables. In our experiments reported in the paper we took $t = 2$. These features are given below. The notation $\langle V_i \rangle_{i \in C}$ stands for average of quantity V_i given by $\frac{1}{|C|} \sum_i V_i$

$$\text{for } t = 1$$

$$f^{(1)}_{e\_prior} = \left\langle P^{(1)}_{e\_prior}(f_{il}) \right\rangle_{1 \le l \le p}$$

$$f^{(1)}_{a\_prior} = \left\langle P^{(1)}_{a\_prior}(pm(f_{il})) \right\rangle_{1 \le l \le p}$$

$$f^{(1)}_{ev\_prior} = \left\langle P^{(1)}_{ev\_prior}(f_{il} \mid v) \right\rangle_{1 \le l \le p}$$

$$f^{(1)}_{av\_prior} = \left\langle P^{(1)}_{av\_prior}(pm(f_{il}) \mid v) \right\rangle_{1 \le l \le p}$$

$$\text{for } t = 2$$

$$f^{(2)}_{e\_prior} = \left\langle P^{(2)}_{e\_prior}(f_{il_1} \wedge f_{il_2}) \right\rangle_{1 \le l_1 < l_2 \le p}$$

$$f^{(2)}_{a\_prior} = \left\langle P^{(2)}_{a\_prior}(pm(f_{il_1} \wedge f_{il_2})) \right\rangle_{1 \le l_1 < l_2 \le p}$$

$$f^{(2)}_{ev\_prior} = \left\langle P^{(2)}_{ev\_prior}(f_{il_1} \wedge f_{il_2} \mid v) \right\rangle_{1 \le l_1 < l_2 \le p}$$

$$f^{(2)}_{av\_prior} = \left\langle P^{(2)}_{av\_prior}(pm(f_{il_1} \wedge f_{il_2}) \mid v) \right\rangle_{1 \le l_1 < l_2 \le p}$$

  The prior tables $P^{(t)}_r(|.)$ are created using the training data.

- *Logical Form and Environment*: As explained in the paper, we introduce the anchored mapping ξ to help in dealing with ellipsis. Therefore, we add a feature that maximizes the similarity between the anchored mapping $\xi(v_j)$ of a variable v_j and the new mapping $\xi_i(v_j)$. This is given by:

$$f_{ee} = \frac{1}{m} \sum_{j=1}^{m} \Delta(\xi(v_j), \xi_i(v_j)))$$

where Δ is a similarity score between objects $\xi(v_j)$ and $\xi_i(v_j)$. We compute this by taking $\Delta(o_1, o_2) = 0.5\ 1\{o_1.\text{category} = o_2.\text{category}\} + 0.5$ fraction of common states value pairs.

- *Relationship Features*: Given all (ω_1, ω_2, r) pairs where $\omega_1, \omega_2 \in c_i$ and r is a spatial relationship between them. The relationship feature is given by:

$$f_{rel} = \langle y_{i,\omega_1,\omega_2} \rangle_{(\omega_1,\omega_2,r)}$$

where $y_{i,\omega_1,\omega_2} = 1$ if post-condition f_i contains a predicate $rel(o_1, o_2)$ where o_1, o_2 are the objects referred by description ω_1, ω_2 respectively.

- *Similarity Feature*: This is given by the Jaccard index of all the words in c_i and the words in the anchored lexical entry.

- *Transition Probabilities*: Given a logical form z_{i-1}, we can set priors on the logical form z_i. E.g., its unlikely that a logical form with post-condition $f_{i-1} = on(cup_1, counter_2)$ will be succeeded by logical form with post-condition $f_i = on(cup_1, counter_1)$. Further, the logical forms that can occur in the end state (c_i is the last frame node) are also restricted. We therefore,

define 3 transition probability features to capture this:

$$f_{tr\_prior} = \left\langle P_{tr\_prior}(f_{i,l_1}, f_{i-1,l_2}) \right\rangle_{l_1,l_2}$$
$$f_{atr\_prior} = \left\langle P_{atr\_prior}(pm(f_{i,l_1}), pm(f_{i-1,l_2})) \right\rangle_{l_1,l_2}$$
$$f_{end} = \left\langle P_{end}(pm(f_{i,l})) \right\rangle_{l}$$

A.7 Assignment Problem

During inference, we want to generate logical forms $z = (\ell, \xi)$ for a given lexical entry $\ell = [\nu \Rightarrow (\lambda\vec{v}.S, \xi')]$. However the number of such logical forms are exponential in the number of variables in $\vec{v}$. Therefore, for practical reasons we only consider the optimum assignment given by $\arg\max_\xi \phi(z = (\ell, \xi), \cdots) \cdot \theta$. Note that we use slightly different notation from the paper, for reasons of brevity.

We convert this assignment problem into an optimization problem and then solve it approximately. To do so, we define 0-1 variables $y_{ij}; 1 \le i \le m; 1 \le j \le n$. Where m is the number of variables in $\vec{v}$ and n is the number of objects in the given environment s. Further $y_{ij} = 1$ iff variable v_i maps to the object o_j. Using this notation, the features described in Section A.6 can be expressed as follows.

1. Language and Logical Form

$$f_{le} = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\max_\omega \rho(\omega, o_j)) y_{ij}$$
$$f_{recall} = \frac{1}{|\omega \in c|} \sum_{\omega \in c} \max_j (\sum_{i=1}^{m} \rho(\omega, o_j) y_{ij})$$

2. Logical Form

The environment prior terms can be easily expressed in a form which is polyno-

mial in y_{ij}. For example, the feature $f^{(2)}_{e\text{-}prior}$ for the parametrized post-condition $(\texttt{state}\ \texttt{v}_1\ \texttt{water}) \wedge (\texttt{on}\ \texttt{v}_1\ \texttt{v}_2)$ can be expressed as:

$$\sum_{r,s=1}^{m} \texttt{P}^{(2)}_{\texttt{e-prior}}((\texttt{state}\ \texttt{o}_\texttt{r}\ \texttt{water}) \wedge (\texttt{on}\ \texttt{o}_\texttt{r}\ \texttt{o}_\texttt{s}))y_{1r}y_{1s}$$

3. Logical Form and Environment

Similarly, the f_{ee} term can be expressed as:

$$f_{ee} = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\Delta(\xi'(v_i), o_j))y_{ij}$$

4. Relationship Feature

For every $(\omega_1, \omega_2, r) \in c$ pair; we find the objects o_{j_1}, o_{j_2} referred to by these descriptions. Let the post-condition f contain atoms $f_1, f_2, \cdots f_l$ of the type $r(v_1, v_2)$ then for each such predicate, we consider the term $y_{1j_1}y_{2j_2}$.

5. Transition Probabilities

Transition probabilities are expressed similar to environment priors.

Dropping the higher order terms (generally small) and the *recall* term(to simplify the optimization); we get a quadratic program of the form:

$$\max a^T x + x^T \mathbf{B} x$$

$$\mathbf{P}x \leq q$$

The linear constraints $\mathbf{P}x \leq q$ consists of $y_{ij} \in \{0, 1\}$, $\sum_j y_{ij} = 1$ and semantic constraints based on preconditions as given in the planner. E.g., for the post-condition $\texttt{on}(\texttt{v}_1, \texttt{v}_2)$, the planner preconditions tells that v_1 mus satisfy $IsGraspable(v_1)$; we therefore add these semantic constraints as inferred from the planner.

In this form, the assignment problem is nonconvex and does not necessarily admit a unique solution. While this can be solved by standard solvers such as AlgLib library; this optimization is quite slow and hence for practical reasons we drop the **B** term and solve the remaining linear program using a fast interior point solver after relaxation. The experiments in the paper are reported based on these approximations.

APPENDIX B

APPENDIX FOR CHAPTER 5

B.1 Reward Shaping Theorems

In Section 5.5, we introduce two reward shaping terms. We follow the safe-shaping theorems of Ng et al. [1999] and Wiewiora et al. [2003]. The theorems outline potential-based terms that realize sufficient conditions for *safe* shaping. Applying safe terms guarantees the order of policies according to the original problem reward does not change. While the theory only applies when optimizing the total reward, we show empirically the effectiveness of the safe shaping terms in a contextual bandit setting. For convenience, we provide the definitions of potential-based shaping terms and the theorems introduced by Ng et al. [1999] and Wiewiora et al. [2003] using our notation. We refer the reader to the original papers for the full details and proofs.

The distance-based shaping term F_1 is defined based on the theorem of Ng et al. [1999]:

Definition. *A shaping term* $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ *is potential-based if there exists a function* $\phi : \mathcal{S} \to \mathbb{R}$ *such that, at time* j, $F(s_j, a_j, s_{j+1}) = \gamma\phi(s_{j+1}) - \phi(s_j)$, $\forall s_j, s_{j+1} \in \mathcal{S}$ *and* $a_j \in \mathcal{A}$, *where* $\gamma \in [0, 1]$ *is a future reward discounting factor. The function* ϕ *is the potential function of the shaping term* F.

Theorem. *Given a reward function* $R(s_j, a_j)$, *if the shaping term is potential-based, the shaped reward* $R_F(s_j, a_j, s_{j+1}) = R(s_j, a_j) + F(s_j, a_j, s_{j+1})$ *does not modify the total order of policies.*

In the definition of F_1, we set the discounting term γ to 1.0 and omit it.

The trajectory-based shaping term F_2 follows the shaping term introduced

by Brys et al. [2015]. To define it, we use the look-back advice shaping term of Wiewiora et al. [2003], who extended the potential-based term of Ng et al. [1999] for terms that consider the previous state and action:

Definition. *A shaping term $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is potential-based if there exists a function $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that, at time j, $F(s_{j-1}, a_{j-1}, s_j, a_j) = \gamma\phi(s_j, a_j) - \phi(s_{j-1}, a_{j-1})$, $\forall s_j, s_{j-1} \in \mathcal{S}$ and $a_j, a_{j-1} \in \mathcal{A}$, where $\gamma \in [0, 1]$ is a future reward discounting factor. The function ϕ is the potential function of the shaping term F.*

Theorem. *Given a reward function $R(s_j, a_j)$, if the shaping term is potential-based, the shaped reward $R_F(s_{j-1}, a_{j-1}, s_j, a_j) = R(s_j, a_j) + F(s_{j-1}, a_{j-1}, s_j, a_j)$ does not modify the total order of policies.*

In the definition of F_2 as well, we set the discounting term γ to 1.0 and omit it.

B.2 Evaluation Systems

We implement multiple systems for evaluation.

STOP The agent performs the `STOP` action immediately at the beginning of execution.

RANDOM The agent samples actions uniformly until `STOP` is sampled or J actions were sampled, where J is the execution horizon.

SUPERVISED Given the training data with N instruction-state-execution triplets, we generate training data of instruction-state-action triplets and optimize the log-likelihood of the data. Formally, we optimize the objective:

$$J = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m^{(i)}} \log \pi(\tilde{s}_j^{(i)}, a_j^{(i)}) \ ,$$

where $m^{(i)}$ is the length of the execution $\bar{e}^{(i)}$, $\tilde{s}_j^{(i)}$ is the agent context at step j in sample i, and $a_j^{(i)}$ is the demonstration action of step j in demonstration execution $\bar{e}^{(i)}$. Agent contexts are generated with the annotated previous actions (i.e., to generate previous images and the previous action). We use minibatch gradient descent with ADAM updates [Kingma and Ba, 2014].

DQN We use deep Q-learning [Mnih et al., 2015] to train a Q-network. We use the architecture described in Section 5.3, except replacing the task specific part with a single 81-dimension layer. In contrast to our probabilistic model, we do not decompose block and direction selection. We use the shaped reward function, including both F_1 and F_2. We use a replay memory of size 2,000 and an ϵ-greedy behavior policy to generate rollouts. We attenuate the value of ϵ from 1 to 0.1 in 100,000 steps and use prioritized sweeping for sampling. We also use a target network that is synchronized after every epoch.

REINFORCE We use the REINFORCE algorithm [Sutton et al., 1999] to train our agent. REINFORCE performs policy gradient learning with total reward accumulated over the roll-out as opposed to using immediate rewards as in our main approach. REINFORCE samples the total reward using monte-carlo sampling by performing a roll-out. We use the shaped reward function, including both F_1 and F_2 terms. Similar to our approach, we initialize with a SUPERVISED model and regularize the objective with the entropy of the policy. We do not use a reward baseline.

SUPERVISED with Oracle Planner We use a variant of our model assuming a perfect planner. The model predicts the block to move and its target position as a pair of coordinates. We modify the architecture in Section 5.3 to predict the block to move and its target position as a pair of coordinates. This model assumes that the sequence of actions is inferred from the predicted target position using an oracle planner. We train using supervised learning by maximizing the likelihood of the block being moved and minimizing the squared distance between the predicted target position and the annotated target position.

B.3 Parameters and Initialization

B.3.1 Architecture Parameters

We use an RGB image of 120x120 pixels, and a convolutional neural network (CNN) with 4 layers. The first two layers apply 32 8×8 filters with a stride of 4, the third applies 32 4×4 filters with a stride of 2. The last layer performs an affine transformation to create a 200-dimension vector. We linearly scale all images to have zero mean and unit norm. We use a single layer RNN with 150-dimensional word embeddings and 250 LSTM units. The dimension of the action embedding ψ_a is 56, including 32 for embedding the block and 24 for embedding the directions. $\mathbf{W}^{(1)}$ is a 506×120 matrix and $\mathbf{b}^{(1)}$ is a 120-dimension vector. $\mathbf{W}^{(D)}$ is 120×20 for 20 blocks, and $\mathbf{W}^{(B)}$ is 120×5 for the four directions (north, south, east, west) and the `STOP` action. We consider $K = 4$ previous images, and use horizon length $J = 40$.

B.3.2 Initialization

Embedding matrices are initialized with a zero-mean unit-variance Gaussian distribution. All biases are initialized to $\mathbf{0}$. We use a zero-mean truncated normal distribution to initialize the CNN filters (0.005 variance) and CNN weights matrices (0.004 variance). All other weight matrices are initialized with a normal distribution (mean=0.0, standard deviation=0.01). The matrices used in the word embedding function ψ are initialized with a zero-mean normal distribution with standard deviation of 1.0. Action embedding matrices, which are used for ψ_a, are initialized with a zero-mean normal distribution with 0.001 standard deviation. We initialize policy gradient learning, including our approach, with parameters estimated using supervised learning for two epochs, except the direction parameters $\mathbf{W}^{(D)}$ and $\mathbf{b}^{(D)}$, which we learn from scratch. We found this initialization method to provide a good balance between strong initialization and not biasing the learning too much, which can result in limited exploration.

B.3.3 Learning Parameters

We use the distance error on a small validation set as stopping criteria. After each epoch, we save the model, and select the final model based on development set performance. While this method overfits the development set, we found it more reliable then using the small validation set alone. Our relatively modest performance degradation on the held-out set illustrates that our models generalize well. We set the reward and shaping penalties $\delta = \delta_f = 0.02$. The entropy regularization coefficient is $\lambda = 0.1$. The learning rate is $\mu = 0.001$ for supervised learning and $\mu = 0.00025$ for policy gradient. We clip the gradient

Name	Size	Vocabulary Size	AIL	# Actions	ATL	Partially Observed
Blocks	16,767	1,426	15.27	81	15.4	No
SAIL	3,237	563	7.96	3	3.12	Yes
Matuszek	217	39	6.65	3	N/A	No
Misra	469	775	48.7	> 100	21.5	No

Table B.1: Comparison of several related natural language instructions corpora. Size denotes the number of instructions in the dataset. AIL is the Average Instruction Length and ATL is the Average Trajectory Length.

at a norm of 5.0. All learning algorithms use a mini-batch of size 32 during training.

B.4 Dataset Comparisons

We briefly review instruction following datasets in Table B.1, including: Blocks [Bisk et al., 2016b], SAIL [MacMahon et al., 2006, Chen and Mooney, 2011], Matuszek [Matuszek et al., 2012b], and Misra [Misra et al., 2015]. Overall, Blocks provides the largest training set and a relatively complex environment with well over 2.43^{18} possible states.[1] The most similar dataset is SAIL, which provides only partial observability of the environment (i.e., the agent observes what is around it only). However, SAIL is less complex on other dimensions related to the instructions, trajectories, and action space. In addition, while Blocks has a large number of possible states, SAIL includes only 400 states. The small number of states makes it difficult to learn vision models that generalize well. Misra [Misra et al., 2015] provides a parameterized action space (e.g., grasp(cup)), which leads to a large number of potential actions. However, the corpus is relatively small.

[1] We compute this loose lower bound on the number of states in the block world as $20! = 2.43^{18}$ (the number of block permutations). This is a very loose lower bound.

B.5 Common Questions

This is a list of potential questions following various decisions that we made. While we ablated and discussed all the crucial decisions in the paper, we decided to include this appendix to provide as much information as possible.

Is it possible to manually engineer a competitive reward function without shaping? Shaping is a principled approach to add information to a problem reward with relatively intuitive potential functions. Our experiments demonstrate its effectiveness. Investing engineering effort in designing a reward function specifically designed to the task is a potential alternative approach.

Are you using beam search? Why not? While using beam search can probably increase our performance, we chose to avoid it. We are motivated by robotic scenarios, where implementing beam search is a challenging task and often not possible. We distinguish between beam search and back-tracking. Beam search is also incompatible with common assumptions of reinforcement learning, although it is often used during test with reinforcement learning systems.

Why are you using the mean of the LSTM hidden states instead of just the final state? We empirically tested both options. Using the mean worked better. This was also observed by Narasimhan et al. [2015]. Understanding in which scenarios one technique is better than the other is an important question for future work.

Can you provide more details about initialization? Please see Appendix B.3.

Does the agent in the block world learn to move obstacles and other blocks? While the agent can move any block at any step, in practice, it rarely happens. The agent prefers to move blocks around obstacles rather than moving other blocks and moving them back into place afterwards. This behavior is learned from the data and shows even when we use only very limited amount of demonstrations. We hypothesize that in other tasks the agent is likely to learn that moving obstacles is advantageous, for example when demonstrations include moving obstacles.

Does the agent explicitly mark where it is in the instruction? We estimate that over 90% of the instructions describe the target position. Therefore, it is often not clear how much of the instruction was completed during the execution. The agent does not have an explicit mechanism to mark portions of the instruction that are complete. We briefly experimented with attention, but found that empirically it does not help in our domain. Designing an architecture to allows such considerations is an important direction for future work.

Does the agent know which blocks are present? Not all blocks are included in each task. The agent must infer which blocks are present from the image and instruction. The set of possible actions, which includes moving all possible blocks, does not change between tasks. If the agent chooses to move a block that is not present, the world state does not change.

Did you experiment with executing sequences of instruction? The Bisk et al. [2016b] includes such instructions, right? The majority of existing corpora, including SAIL [Chen and Mooney, 2011, Artzi and Zettlemoyer, 2013, Mei

et al., 2016a], provide segmented sequences of instructions. Existing approaches take advantage of this segmentation during training. For example, Chen and Mooney [2011], Artzi and Zettlemoyer [2013], and Mei et al. [2016a] all train on segmented data and test on sequences of instructions by doing inference on one sentence at a time. We are also able to do this. Similar to these approaches, we will likely suffer from cascading errors. The multi-instruction paragraphs in the Bisk et al. [2016b] data are an open problem and present new challenges beyond just instruction length. For example, they often merge multiple block placements in one instruction (e.g, *put the SRI, HP, and Dell blocks in a row*). Since the original corpus does not provide trajectories and our automatic generation procedure is not able to resolve which block to move first, we do not have demonstrations for this data. The instructions also present a significantly more complex task. This is an important direction for future work, which illustrates the complexity and potential of the domain.

Potential-based shaping was proven to be safe when maximizing the total expected reward. Does this apply for the contextual bandit setting, where you maximize the immediate reward? The safe shaping theorems (Appendix B.1) do not hold in our contextual bandit setting. We show empirically that shaping works in practice. However, how and if it changes the order of policies is an open question.

How long does it take to train? How many frames the agent observes? The agent observes about 2.5 million frames. It takes 16 hours using 50% capacity of an Nvidia Pascal Titan X GPU to train using our approach. DQN takes more than twice the time for the same number of epochs. Supervised learning takes

about 9 hours to converge. We also trained DQN for around four days, but did not observe improvement.

Did you consider initializing DQN with supervised learning? Initializing DQN with the probabilistic supervised model is challenging. Since DQN is not probabilistic it is not clear what this initialization means. Smart initialization of DQN is an important problem for future work.

APPENDIX C

APPENDIX FOR CHAPTER 6

C.1 Tasks and Data: Comparisons

Table C.1 provides summary statistics comparing LANI and CHAI to existing related resources.

C.2 Reward Function

LANI Following Misra et al. [2017], we use a shaped reward function that rewards the agent for moving towards the goal location. The reward function for example i is:

$$R^{(i)}(s, a, s') = R_p^{(i)} + \phi^{(i)}(s) - \phi^{(i)}(s')$$

where s' is the origin state, a is the action, s is the target state, $R_p^{(i)}$ is the problem reward, and $\phi^{(i)}(s) - \phi^{(i)}$ is a shaping term. We use a potential-based shaping [Ng et al., 1999] that encourages the agent to both move and turn towards the goal.

Dataset	Size	Vocabulary Size	AIL	Action Space Size	ATL	Partially Observed
Bisk et al. [2016a]	16,767	1,426	15.27	81	15.4	No
MacMahon et al. [2006]	3,237	563	7.96	3	3.12	Yes
Matuszek et al. [2012b]	217	39	6.65	3	N/A	No
Misra et al. [2015]	469	775	48.7	>100	21.5	No
LANI	28,204	2,292	12.07	4	24.6	Yes
CHAI	13,729	1018	10.14	1028	54.5	Yes

Table C.1: Comparison of LANI and CHAI to several existing natural language instructions corpora. Size denotes the number of instructions in the dataset. AIL is the Average Instruction Length and ATL is the Average Trajectory Length.

The potential function is:

$$\phi^{(i)}(s) = \delta \textsc{TurnDist}(s, s_g^{(i)}) + (1 - \delta)\textsc{MoveDist}(s, s_g^{(i)}) \ ,$$

where MoveDist is the euclidean distance to the goal normalized by the agent's forward movement distance, and TurnDist is the angle the agent needs to turn to face the goal normalized by the agent's turn angle. We use δ as a gating term, which is 0 when the agent is near the goal and increases monotonically towards 1 the further the agent is from the goal. This decreases the sensitivity of the potential function to the TurnDist term close to the goal. The problem reward $R_p^{(i)}$ provides a negative reward of up to -1 on collision with any object or boundary (based on the angle and magnitude of collision), a negative reward of -0.005 on every action to discourage long trajectories, a negative reward of -1 on an unsuccessful stop, when the distance to the goal location is greater than 5, and a positive reward of +1 on a successful stop.

Chai We use a similar potential based reward function as Lani. Instead of rewarding the agent to move towards the final goal the model is rewarded for moving towards the next intermediate goal. We heuristically generate intermediate goals from the human demonstration by generating goals for objects to be interacted with, doors that the agent should enter, and the final position of the agent. The potential function is:

$$\phi^{(i)}(s) = \textsc{TurnDist}(s, s_{g,j}^{(i)}) + \textsc{MoveDist}(s, s_{g,j}^{(i)}) + \textsc{IntDist}(s, s_{g,j}^{(i)}) \ ,$$

where $s_{g,j}^{(i)}$ is the next intermediate goal, TurnDist rewards the agent for turning towards the goal, MoveDist rewards the agent for moving closer to the goal, and IntDist rewards the agent for accomplishing the interaction in the

intermediate goal. The goal is updated on being accomplished. Besides the potential term, we use a problem reward $R_p^{(i)}$ that gives a reward of 1 for stopping near a goal, -1 for colliding with obstacles, and -0.002 as a verbosity penalty for each step.

C.3 Baseline Details

MISRA17 We use the model of Misra et al. [2017]. The model uses a convolution neural network for encoding the visual observations, a recurrent neural network with LSTM units to encode the instruction, and a feed-forward network to generate actions using these encodings. The model is trained using policy gradient in a contextual bandit setting. We use the code provided by the authors.

CHAPLOT18 We use the gated attention architecture of Chaplot et al. [2018]. The model is trained using policy gradient with generalized advantage estimation [Schulman et al., 2015b]. We use the code provided by the authors.

Our Approach with Joint Training We train the full model with policy gradient. We maximize the expected reward objective with entropy regularization. Given a sampled goal location $l_g \sim p(. \mid \bar{x}, I_P)$ and a sampled action $a \sim p(. \mid l_g, (I_1, p_1), \ldots, (I_t, p_t))$, the update is:

$$\nabla J \approx \Big\{\nabla \log P(l_g \mid \bar{x}, I_P) + \nabla \log P(a_t \mid l_g, (I_1, p_1), \ldots, (I_t, p_t))\Big\} R(s_t, a) + \lambda \nabla H(\pi(. \mid \tilde{s}_t) \quad .$$

We perform joint training with randomly initialized goal prediction and action

generation models.

C.4 Hyperparameters

For LANI experiments, we use 5% of the training data for tuning the hyperparameters and train on the remaining. For CHAI, we use the development set for tuning the hyperparameters. We train our models for 20 epochs and find the optimal stopping epoch using the tuning set. We use 32 dimensional embeddings for words and time. LSTM_x and LSTM_A are single layer LSTMs with 256 hidden units. The first layer of CNN_0 contains 128 8×8 kernels with a stride of 4 and padding 3, and the second layer contains 64 3×3 kernels with a stride of 1 and padding 1. The convolution layers in LINGUNETuse 32 5×5 kernels with stride 2. All deconvolutions except the final one, also use 32 5×5 kernels with stride 2. The dropout probability in LINGUNETis 0.5. The size of attention mask is 32×32 + 1. For both LANI and CHAI, we use a camera angle of 60° and create panoramas using 6 separate RGB images. Each image is of size 128×128. We use a learning rate of 0.00025 and entropy coefficient λ of 0.05.

C.5 CHAI Error Analysis

Table C.2 provides the same kind of error analysis results here for the CHAI dataset as we produced for LANI, comparing performance of the model on samples of sentences with and without the analysis phenomena that occurred in CHAI.

Category	Present	Absent	p-value
Spatial relations	2.56	1.77	.023
Location conjunction	3.85	1.93	.226
Temporal coordination	1.70	2.14	.164
Co-reference	1.98	1.98	.993

Table C.2: Mean goal prediction error for CHAI instructions with and without the analysis categories we used in Table 6.2. The p-values are from two-sided t-tests comparing the means in each row.

C.6 Examples of Generated Goal Prediction

Figure C.1 shows example goal predictions from the development sets. We found the predicted probability distributions to be reasonable even in many cases where the agent failed to successfully complete the task. We observed that often the evaluation metric is too strict for LANI instructions, especially in cases of instruction ambiguity.

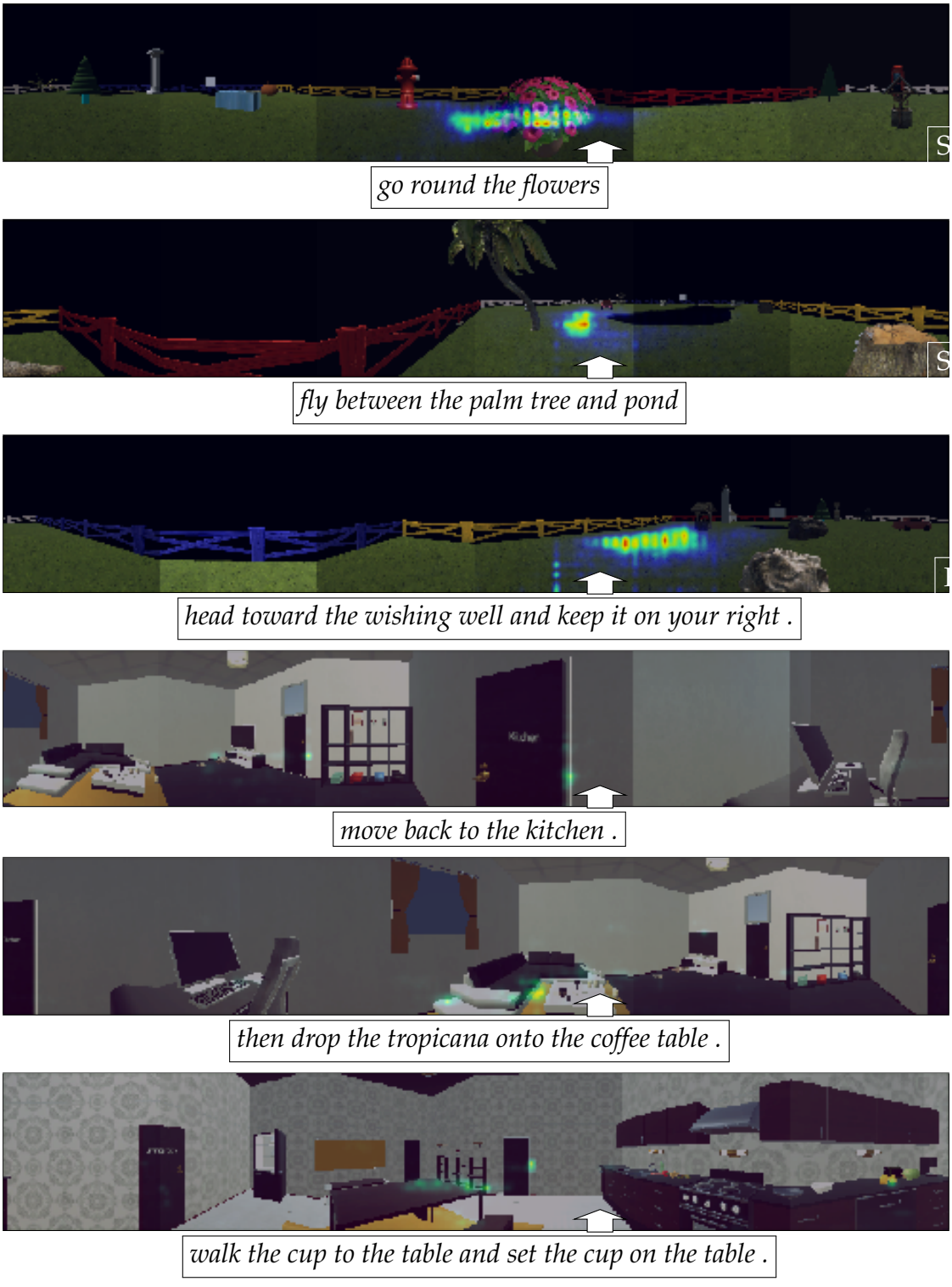


Figure C.1: Goal prediction probability maps P_g overlaid on the corresponding observed panoramas I_P. The top three examples show results from LANI, the bottom three from CHAI. The white arrow indicates the forward direction that the agent is facing. The success/failure in the LANI examples indicate if the task was completed accurately or not following the task completion (TC) metric.

BIBLIOGRAPHY

Google image search. http://www.images.google.com/.

Iq engines: Image recognition apis for photo albums and mobile commerce. https://www.iqengines.com/.

List of drone companies. https://uavcoach.com/drone-companies/.

Pytorch. https://pytorch.org/.

Alekh Agarwal, Daniel J. Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *Proceedings of the International Conference on Machine Learning*, 2014.

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 *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the annual meeting on Association for Computational Linguistics*, 2016.

Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

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

Yoav Artzi and Luke S. Zettlemoyer. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

Yoav Artzi, Dipanjan Das, and Slav Petrov. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.

Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 264–273, 2018.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002a.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002b.

Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Association for Computational Linguistics (ACL)*, 2014.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Yonatan Bisk, Daniel Marcu, and William Wong. Towards a dataset for human computer communication via grounded language acquisition. In *Proceedings of the AAAI Workshop on Symbiotic Cognitive Systems*, 2016a.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. Natural language communication with robots. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016b.

Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A. Knepper, and Yoav Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. In *Proceedings of the Robotics: Science and Systems Conference*, 2018a.

Valts Blukis, Dipendra Misra, Ross A. Knepper, and Yoav Artzi. Mapping navigation instructions to continuous control actions with position visitation prediction. In *Proceedings of the Conference on Robot Learning*, 2018b.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

SRK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 82–90, 2009.

SRK Branavan, Luke S Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL)*, pages 1268–1277, 2010.

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2015.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. 2018.

David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 859–865, 2011.

Howard Chen, Alane Suhr, Dipendra Misra, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Conference on Computer Vision and Pattern Recognition*, 2019.

Istvan Chung, Oron Propp, Matthew R Walter, and Thomas M Howard. On the performance of hierarchical distributed correspondence graphs for efficient symbol grounding of robot instructions. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5247–5252. IEEE, 2015.

Andrea F Daniele, Mohit Bansal, and Matthew R Walter. Navigational instruction generation as inverse reinforcement learning with neural machine translation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pages 109–118. IEEE, 2017.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–38, 1977.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Simon S Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudík, and John Langford. Provably efficient rl with rich observations via latent state decoding. *arXiv preprint arXiv:1901.09018*, 2019.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Conference on Learning Theory (COLT)*, 2010.

Felix Duvallet, Matthew R Walter, Thomas Howard, Sachithra Hemachandra, Jean Oh, Seth Teller, Nicholas Roy, and Anthony Stentz. Inferring maps and behaviors from natural language instructions. In *Experimental Robotics*, pages 373–388. Springer, 2016.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

Juan Fasola and Maja J Mataric. Interpreting instruction sequences in spatial language discourse with pragmatics towards natural human-robot interaction. In *International Conference on Robotics and Automation (ICRA)*, pages 6667–6672, 2014.

Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

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. *CoRR*, abs/1806.02724, 2018.

Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Sergio Guadarrama, Lorenzo Riano, Dave Golland, Daniel Go, Yangqing Jia, Dan Klein, Pieter Abbeel, and Trevor Darrell. Grounding spatial relations for human-robot interaction. In *International Conference on Intelligent Robots and Systems (IROS)*, 2013.

Sergio Guadarrama, Erik Rodner, Kate Saenko, Ning Zhang, Ryan Farrell, Jeff Donahue, and Trevor Darrell. Open-vocabulary object retrieval. In *Robotics: science and systems*, volume 2, page 6. Citeseer, 2014.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014.

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.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. Grounded language learning in a simulated 3D world. *CoRR*, abs/1706.06551, 2017.

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. *arXiv preprint arXiv:1710.02298*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9, 1997.

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, 2014.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6, 2018.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Machine Learning, Proceedings of the Nineteenth International Conference , University of New South Wales, Sydney, Australia, July 8-12, 2002*, 2002.

Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara L. Berg. Referitgame: Referring to objects in photographs of natural scenes. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proeceediings of the International Joint Conference on Artificial Intelligence*, 1999.

Joohyun Kim and Raymond Mooney. Unsupervised PCFG induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2014.

Nikita Kitaev and Dan Klein. Where is misty? interpreting spatial descriptors by modeling regions in space. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2017.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Association for Computational Linguistics (ACL)*, pages 423–430, 2003.

Chen Kong, Dahua Lin, Mohit Bansal, Raquel Urtasun, and Sanja Fidler. What are you talking about? text-to-image coreference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

Akshay Krishnamurthy, Alekh Agarwal, and Miro Dudik. Contextual semibandits via supervised learning oracles. In *Advances In Neural Information Processing Systems*, pages 2388–2396, 2016a.

Akshay Krishnamurthy, Alekh Agarwal, and John Langford. PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, 2016b.

Jayant Krishnamurthy and Thomas Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, 1, 2013.

Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2010.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in CCG grammar induction for semantic parsing. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*, pages 817–824, 2008.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17, 2016.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches

to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013.

James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda Muresan, S Bertel Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding english commands to reward functions. In *Robotics: Science and Systems*, 2015.

Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *National Conference on Artificial Intelligence*, 2006.

Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315. IEEE, 2010.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan Yuille, and Kevin Murphy. Generation and Comprehension of Unambiguous Object Descriptions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016. doi: 10.1109/CVPR.2016.9.

Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *Proceedings of the international conference on Human-robot interaction*, 2010.

Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. A joint model of language and perception for grounded attribute learning. In *International Conference on Machine Learning (ICML)*, 2012a.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *International Symposium on Experimental Robotics (ISER)*, 2012b.

Hongyuan Mei, Mohit Bansal, and Matthew Walter. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016a.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016b.

Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2017.

Dipendra Kumar Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell Me Dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems (RSS)*, 2014.

Dipendra Kumar Misra, Kejia Tao, Percy Liang, and Ashutosh Saxena. Environment-driven lexicon induction for high-level instructions. In *Association for Computational Linguistics (ACL)*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with

deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2013.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, 1999.

Khanh Nguyen, Debadeepta Dey, Chris Brockett, and Bill Dolan. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. *arXiv preprint arXiv:1812.04155*, 2018.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29:19–51, 2003.

Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *Proceedings of the International Conference on Machine Learning*, 2016.

Junhyuk Oh, Satinder P. Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the international conference on machine learning*, 2017.

Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. *arXiv preprint arXiv:1704.03084*, 2017.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21, 2008.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 693–701, 2011.

Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193, 2012.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation

learning and structured prediction to no-regret online learning. In *Artificial Intelligence and Statistics (AISTATS)*, 2011.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *CoRR*, 2016.

Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2): 157–173, 2008.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015b.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

Mark Steedman and Jason Baldridge. Combinatory categorial grammar. *Non-Transformational Syntax*, pages 181–224, 2003.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888. ACM, 2006.

Pei-Hao Su, Milica Gasic, Nikola Mrkšić, Lina M. Rojas Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. On-line active reward learning for policy optimisation in spoken dialogue systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

Alane Suhr and Yoav Artzi. Situated mapping of sequential instructions to actions with single-step reward observation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2018.

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

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 1999.

Stefanie Tellex and Deb Roy. Grounding spatial prepositions for video search. In *International Conference on Multimodal Interfaces (ICMI)*, pages 253–260, 2009.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.

Stefanie Tellex, Ross Knepper, Adrian Li, Daniela Rus, and Nicholas Roy. Asking for help using inverse semantics. In *Proceedings of the Robotics: Science and Systems Conference*, 2014.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

Adam Vogel and Daniel Jurafsky. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL)*, pages 806–814, 2010.

Matthew R Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth Teller. A framework for learning semantic maps from grounded natural language descriptions. *The International Journal of Robotics Research*, 33 (9):1167–1190, 2014.

Eric Wiewiora, Garrison W. Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the International Conference on Machine Learning*, 2003.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 1992.

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1): 1–191, 1972.

Wenhan Xiong, Xiaoxiao Guo, Mo Yu, Shiyu Chang, Bowen Zhou, and William Yang Wang. Scheduled policy optimization for natural language communication with intelligent agents. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2018.

Claudia Yan, Dipendra Kumar Misra, Andrew Bennett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. Chalet: Cornell house agent learning environment. *CoRR*, abs/1801.07357, 2018.

Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, 2007.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.