BROWN UNIVERSITY

HONORS THESIS

Grounding Natural Language to Goals for Abstraction, Generalization, and Interpretability

---

Author:
Siddharth KARAMCHETI

Advisors:
Dr. Eugene CHARNIAK
Dr. Stefanie TELLEX

Reader:
Dr. George KONIDARIS

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science with Honors

April 15, 2018

Abstract

Language is universal and powerful: it allows people to communicate their ideas, telegraph their intents, and interact with the world around them in an incredibly natural way. Therefore, as new technology is developed and integrated into everyday life, it is necessary to factor in language, and build interfaces that bridge the gap between human users and arbitrarily complex systems. Language grounding is an instrumental part of this process, defined as the problem of mapping natural language to behavior on an underlying system or environment. These systems and environments can take many forms, including physical systems like robots (mapping language to low-level actuators), software systems like databases and constraint solvers (translating language to programs and structured queries), or environments like real-world landscapes and photographic images (bridging language and perception).

In this thesis, we examine the general problem of language grounding as it relates to two separate and important application areas: the first is the problem of grounding in the context of human-robot interaction, while the second is the problem of grounding in the context of question-answering on short stories. We use these two application areas to guide our exploration of three different themes in language as a whole, and how they manifest themselves in the problem of language grounding: 1) **abstraction**, or how to interpret language of varying complexities, 2) **generalization**, or how to extrapolate and map language to new behaviors, and 3) **interpretability**, or how to reason about language in an understandable way.

Each chapter of this thesis explores one of the aforementioned themes in greater detail and motivates its utility before surveying related approaches and evaluating a proposed, novel approach that better addresses the theme in question. Building on techniques from deep learning, natural language processing, reinforcement learning, and planning, we introduce a full suite of diverse models for language grounding, each demonstrating state-of-the-art performance on tasks spanning human-robot interaction and question-answering. Finally, we discuss next steps, describing the limitations of the current work in language grounding before outlining several important open problems that motivate future research directions.

Acknowledgements

First and foremost, I want to thank my advisors, Professors Eugene Charniak and Stefanie Tellex: My love of research, my passion for language, and all the work contained in this thesis are a result of their mentorship. Thank you so much for taking a chance on me, and instilling in me the confidence and excitement for language and robots you carry with you every day. Thank you for the random chats in the lab or your office about my latest crazy ideas, the countless hours editing and re-editing before paper deadlines, and for everything else that I'm forgetting – I'll forever be grateful.

To my reader, Professor George Konidaris: thank you so much for all the random office chats, answering my late-night emails asking for advice, and for forcing me to think about the big questions – about what it really means to work in AI.

To my collaborators and friends, Dilip, Nakul, Lawson, Eddie, and Mina: Thanks so much for helping me succeed – for the countless hours spent collecting data, troubleshooting deep nets, and getting tables to fit on one page the hour before a paper deadline. Nakul and Lawson, thank you so much for your wisdom and your mentorship, and for teaching me the ins and outs of the research process. Eddie and Mina, thank you for all your help throughout this last year. And last, but definitely not least, Dilip – thank you brother, for the countless late nights writing and hacking, for always grounding me and giving perspective, and for always being willing to go out and see a movie when I need it the most.

To my dear friends, at Brown and back home. Arun, Helen, Julie, Victoria, and everyone else (you know who you are) – thank you for always standing by me, and for stubbornly resisting my desire to stay inside and work, pulling me out into the world for some truly unforgettable experiences. Thank you for always being willing to talk about the little and the not-so-little things, for the late night car rides, and for the much-needed reality checks – I'm so incredibly lucky and grateful to have you all in my life.

Lastly, to my family. Sanjana and Meena, thank you for being the best sisters I could have asked for – you two shaped me into the person I am today. And finally, to my parents: Amma and Nana, there aren't enough words to do this justice, so I'll just say this: thank you for always loving and supporting me, through good, bad, and everything in between.

Contents

List of Figures

List of Tables

Chapter 1

A Grounded Introduction

Technology is everywhere: we live in a world of self-driving cars, household robotics, and smart gadgets – technology that has been deeply integrated into our lives. In the years to come, these advancements will only continue, and we will see more and more solutions to countless important problems spanning application areas in manufacturing, healthcare, transportation, and education, amongst others. However, to fully realize and appreciate the magnitude of these advancements, the underlying technologies must be **accessible**, so that they can be used by everyday people.

At the core of realizing true accessibility is language. Language is both universal and extraordinarily powerful – it allows people to communicate their ideas with others and interact with the world around them in a natural way. Therefore, it is necessary to build powerful tools and natural language interfaces, capable of bridging the gap between everyday users and these arbitrarily complex technological systems.

In this thesis, we build these natural language interfaces through examination of language grounding, or the problem of mapping language to behaviors on an underlying system or environment, as it relates to two separate and important application areas: the first is the problem of grounding in the context of human-robot interaction, while the second is the problem of grounding in the context of question-answering on short stories. We use these two application areas to guide our exploration of three different themes in language as a whole, and how they manifest themselves in the problem of language grounding: 1) **abstraction**, or how to interpret language of varying complexities, 2) **generalization**, or how to extrapolate and map language to new behaviors, and 3) **interpretability**, or how to reason about language in an understandable way.

In the first chapter, we tackle **abstraction** through the lens of hierarchical language grounding for robotics. Here, the key realization is that users might give natural language instructions that exist at different complexities: someone might give a general, high-level instruction like "Take the chair to the kitchen," or a low-level, granular instruction like "Take three steps forward, turn left, raise your arm six inches, and turn the doorknob." To accurately and efficiently map to the correct behavior, a language grounding system needs to take this explicit

abstraction into account, and use the hierarchy informed by language to improve reliability. We present a series of models for tackling this problem, and show that we can significantly improve upon prior language grounding work, grounding natural language instructions from real-world users to robot behavior in a fast and accurate manner.

The second chapter examines **generalization**, also in application to robotics. Here, the key realization is that there is a difference in how users specify behavior: someone give an instruction that either specifies a goal condition (or some target state), or they could provide an instruction that specifies a set sequence of actions to take. Generalizing across actions and goals is incredibly important, as in some cases, it matters exactly how a robot agent performs a task, while in others, it may only serve to slow down the grounding process. To this end, we present a new suite of grounding models in tandem with a new, modular representation for language. Our results show that not only can our grounding models more accurately ground action-oriented and goal-oriented commands, but that our models can also generalize effectively, and map language to new tasks that the models have never seen before.

Finally, the third chapter examines **interpretability**, through application short story understanding. The key realization here is that in many existing models for question-answering, the reasoning necessary for answering complex questions about short stories or news articles happens in an opaque, unintuitive form, inside of a deep neural network model. This is a huge problem, in case we want users to be able to trust, question, and interact with these models. To this end, we reformulate the problem of question-answering as one of language grounding, where we learn to map sentences of stories to updates on a human-readable database. We then answer questions by querying this database, allowing human users insight into the reasoning process. Our results show that our models are not only capable of superior performance, but that they also require significantly less data than existing state-of-the-art techniques for question-answering.

All together, the goal of this thesis is to expand upon the state-of-the-art and introduce new problems and initial solutions for several important areas of language grounding. By thinking about these problems, the hope is that we can build more robust, accurate, and efficient language grounding systems, closing the accessibility gap between everyday users and complex technology.

Chapter 2

Hierarchical Language Grounding

This chapter consists of work that was done jointly with Dilip Arumugam, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex[1], presented as a conference paper at Robotics: Science and Systems 2017.

The central theme of this chapter is **abstraction**. We present a series of methods for interpreting language instructions that exist at varying levels of complexity; specifically, we show how to leverage the resulting task hierarchy to obtain more accurate and efficient grounding.

2.1 Abstract

Humans can ground natural language commands to tasks at both abstract and fine-grained levels of specificity. For instance, a human forklift operator can be instructed to perform a high-level action, like "grab a pallet" or a low-level action like "tilt back a little bit." While robots are also capable of grounding language commands to tasks, previous methods implicitly assume that all commands and tasks reside at a single, fixed level of abstraction. Additionally, methods that do not use multiple levels of abstraction encounter inefficient planning and execution times as they solve tasks at a single level of abstraction with large, intractable state-action spaces closely resembling real world complexity. In this work, by grounding commands to all the tasks or subtasks available in a hierarchical planning framework, we arrive at a model capable of interpreting language at multiple levels of specificity ranging from coarse to more granular. We show that the accuracy of the grounding procedure is improved when simultaneously inferring the degree of abstraction in language used to communicate the task. Leveraging hierarchy also improves efficiency: our proposed approach enables a robot to respond to a command within one second on 90% of our tasks, while baselines take over twenty seconds on half the tasks. Finally, we demonstrate that a real, physical robot can ground commands at multiple levels of abstraction allowing it to efficiently plan different subtasks within the same planning hierarchy.

[1]Dilip Arumugam*, Siddharth Karamcheti*, Nakul Gopalan, Lawson L. S. Wong, Stefanie Tellex: "Accurately and Efficiently Interpreting Human-Robot Instructions of Varying Granularities". *Robotics: Science and Systems 2017*

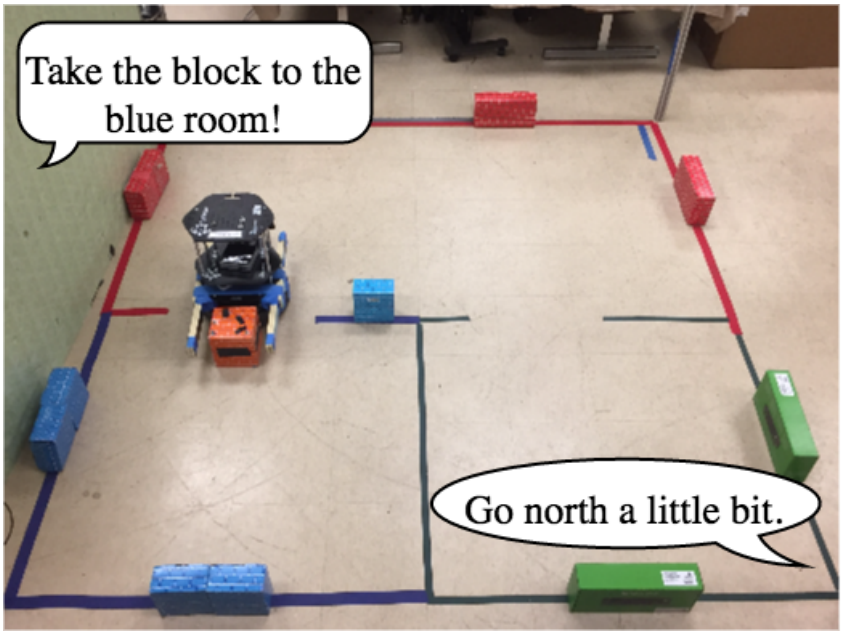


FIGURE 2.1: Examples of high-level and fine-grained commands issued to the Turtlebot robot in a mobile-manipulation task.

2.2 Introduction

In everyday speech, humans use language at multiple levels of abstraction. For example, a brief transcript from an expert human forklift operator instructing a human trainee has very abstract commands such as "Grab a pallet," mid-level commands such as "Make sure your forks are centered," and very fine-grained commands such as "Tilt back a little bit" all within thirty seconds of dialog. Humans use these varied granularities to specify and reason about a large variety of tasks with a wide range of difficulties. Furthermore, these abstractions in language map to subgoals that are useful when interpreting and executing a task. In the case of the forklift trainee above, the sub-goals of moving to the pallet, placing the forks under the object, then lifting it up are all implicitly encoded in the command "Grab a pallet." By decomposing generic, abstract commands into modular sub-goals, humans exert more organization, efficiency, and control in their planning and execution of tasks. A robotic system that can identify and leverage the degree of specificity used to communicate instructions would be more accurate in its task grounding and more robust towards varied human communication.

Existing approaches map between natural language commands and a formal representation at some fixed level of abstraction (Chen and Mooney, 2011; Matuszek et al., 2012; Tellex et al., 2011). While effective at directing robots to complete predefined tasks, mapping to fixed sequences of robot actions is unreliable in changing or stochastic environments. Accordingly, MacGlashan et al.

(2015) decouple the problem and use a statistical language model to map between language and robot goals, expressed as reward functions in a Markov Decision Process (MDP). Then, an arbitrary planner solves the MDP, resolving any environment-specific challenges with execution. As a result, the learned language model can transfer to other robots with different action sets so long as there is consistency in the task representation (*i.e.*, reward functions). However, MDPs for complex, real-world environments face an inherent tradeoff between including low-level task representations and increasing the time needed to plan in the presence of both low- and high-level reward functions (Gopalan et al., 2017).

To address these problems, we present an approach for mapping natural language commands of varying complexities to reward functions at different levels of abstraction within a hierarchical planning framework. This approach enables the system to quickly and accurately interpret both abstract and fine-grained commands. Our system uses a deep neural network language model that maps natural language commands to the appropriate level of the planning hierarchy. By coupling abstraction-level inference with the overall grounding problem, we exploit the subsequent hierarchical planner to efficiently execute the grounded tasks. To our knowledge, we are the first to contribute a system for grounding language at multiple levels of abstraction, as well as the first to contribute a deep learning system for improved robotic language understanding.

Our evaluation shows that deep neural network language models can infer reward functions more accurately than statistical language model baselines. We present results comparing a traditional statistical language model to three different neural architectures that are commonly used in natural language processing. Furthermore, we show that a hierarchical approach allows the planner to map to a larger, richer space of reward functions more quickly and more accurately than non-hierarchical baselines. This speedup allows the robot to respond faster and more accurately to a user's request, with a much larger set of potential commands than previous approaches. We also demonstrate on a Turtlebot the rapid and accurate response of our system to natural language commands at varying levels of abstraction.

2.3 Related Work

Humans use natural language to communicate ideas, motivations, task descriptions, etc. with other humans. Some of the earliest works in this area mapped tasks to another planning language, which then grounded to the actions performed by the robots (Dzifcak et al., 2009a; Chen and Mooney, 2011). More recent methods ground natural language commands to tasks using features that describe correspondences between natural language phrases present in the task

description to the physical objects (Howard, Tellex, and Roy, 2014; Matuszek et al., 2012; Tellex et al., 2011; Brooks et al., 2012; Raman and Kress-Gazit, 2011), or abstract spatial concepts Paul et al., 2016, present in the world and the actions available in the world. This featurized representation can then describe the sequence of actions needed to complete the task. All these approaches ground commands to action sequences, leading to brittle behavior if the environment is stochastic.

MacGlashan et al. (2015) proposed grounding natural language commands to reward functions associated with certain tasks, allowing robot agents to plan in stochastic environments. They treat the goal reward function as a sequence of propositional functions, much like a machine language, to which a natural language task can be translated, using an IBM Model 2 Brown et al., 1990; Brown et al., 1993 (IBM2) language model. While their propositional functions only lie at one level of abstraction, we want the robot to understand commands at different levels of specificity while still maintaining efficient planning and execution in the face of multiple levels of abstraction.

Crucially, MacGlashan et al. (2015) actually perform inference over reward function templates, or lifted reward functions, along with environmental constraints. A lifted reward function merely specifies a task while leaving the environment-specific variables of the task undefined. The environmental binding constraints then specify the properties that an object in the environment must satisfy in order to be bound to a lifted reward function variable. By doing this, the output space of the language model is never tied to any particular instantiation of the environment, but can instead align to objects and attributes that lie within some distribution over environments. Given a lifted reward function and environment constraints (henceforth jointly referred to as only a lifted reward function), a subsequent model can later infer the environment-specific variables without needing to relearn the language understanding components for each environment. In order to leverage this flexibility, all of our proposed language models produce lifted reward functions which are then completed by a grounding module before being passed to the planner (see Sec. 2.5).

Planning in domains with large state-action spaces is computationally expensive as planners like value iteration and bounded real-time dynamic programming (RTDP) need to explore the domain at the lowest, "flat" level of abstraction (Bellman, 1957b; McMahan, Likhachev, and Gordon, 2005). Naively this might result in an exhaustive search of the space before the goal state is found. A better approach is to decompose the planning problem into smaller, more easily solved subtasks. The agent can then achieve the goal by choosing a sequence of these subtasks. A common method to describe subtasks is by using temporal abstraction in the form of macro-actions (McGovern, Sutton, and Fagg, 1997) or options (Sutton, Precup, and Singh, 1999). These methods achieve subgoals using either a fixed sequence of actions (McGovern, Sutton, and Fagg, 1997) or a

subgoal based policy (Sutton, Precup, and Singh, 1999). Planning with macro-actions or options requires computing the policies for each option or macro-action, which is done by exploring and backing up rewards from lowest level actions. This "bottom-up" planning is slow, as the reward for each action taken needs to be backed up through the hierarchy of options, which is time consuming. Other methods for abstraction, like MAXQ (Dieterrich, 2000), R-MAXQ (Jong and Stone, 2008) and Abstract Markov Decision Processes (AMDPs) (Gopalan et al., 2017) involve providing a hierarchy of subtasks. In these methods, a subtask is associated with a subgoal and a state abstraction relevant to achieving the subgoal (Dieterrich, 2000; Gopalan et al., 2017; Jong and Stone, 2008). Both MAXQ (Dieterrich, 2000) and R-MAXQ (Jong and Stone, 2008) are bottom-up planners, they back up each individual action's reward across the hierarchy.

We use AMDPs (Gopalan et al., 2017) in this paper because they plan in a "top-down" fashion. AMDPs offer model-based hierarchical representations in the form of reward functions and transition functions to every subtask. An AMDP hierarchy itself is an acyclic graph in which each node is a primitive action or an AMDP that solves a subtask defined by its parent; the states of each subtask AMDP are abstract representations of the environment state. AMDPs have been shown to achieve faster planning performance than other hierarchical methods Gopalan et al., 2017.

We use a deep neural network language model to perform language grounding. Deep neural networks have had great success in many natural language processing (NLP) tasks, such as traditional language modeling (Bengio et al., 2000; Mikolov et al., 2010; Mikolov et al., 2011), machine translation (Cho et al., 2014; Chung et al., 2014), and text categorization (Iyyer et al., 2015). One reason for their success is the ability to learn meaningful input representations (Bengio et al., 2000; Mikolov et al., 2013). These "embeddings" are dense vectors that not only uniquely represent individual words (as opposed to otherwise sparse approaches for word representation), but also capture semantically significant features of the language. Another reason is the use of recurrent neural networks (RNNs), a type of neural network cell that maps variable length inputs (i.e. commands) to a fixed-size vector representation, which have been widely used in NLP (Cho et al., 2014; Chung et al., 2014; Yamada et al., 2016). Our approach uses both word embeddings and a state-of-the-art RNN model to map between natural language and MDP reward functions.

2.4 Approach

To interpret a variety of natural language commands, there must be a representation for all possible tasks and subtasks. We specify an *Object-oriented Markov*

Decision Process (OO-MDP) to model the robot's environment and actions (Diuk, Cohen, and Littman, 2008). An MDP is a five-tuple of $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ represents the set of states that define an environment, $\mathcal{A}$ denotes the set of actions an agent can execute to transition between states, $\mathcal{T}$ defines the transition probability distribution over all possible next states given a current state and executed action, $\mathcal{R}$ defines the numerical reward earned for a particular transition, and γ represents the discount factor or effective time horizon under consideration. Planning in an MDP produces a mapping between states and actions, or policy, that maximizes the total expected discounted reward. In our framework, as in MacGlashan et al. (2015), we will map between words in language and specific reward functions.

An OO-MDP builds upon an MDP by adding sets of object classes and propositional functions; each object class is defined by a set of attributes and each propositional function is parameterized by instances of object classes. For example, an OO-MDP for the mobile robot manipulation domain seen in Fig. 2.1 might denote the robot's successful placement of the orange block into the blue room via the propositional function blockInRoom block0 room1, where block0 and room1 are instances of the block and room object classes respectively and the blockInRoom propositional function checks if the location attribute of block0 is contained in room1. Using these propositional functions as reward functions that encode termination conditions for each task, we arrive at a sufficient, semantic representation for grounding language. For our evaluation, we use the Cleanup World (Junghanns and Schaeeer, 1997; MacGlashan et al., 2015) OO-MDP, which models a mobile manipulator robot; this domain is defined in Sec. 2.6.1.

However, this approach does not generalize well to different environment configurations. At training time, any natural language command that moves objects or agents to a specific room is conditioned to map room attributes to specific room instances (i.e. in the case of Fig. 2.1, the blue room is always room1). With this in mind, consider what happens if we switched the blue and green rooms at test time, so that the green room is now room1. In this case, any language command that moves an object or agent to the blue room would fail, as the room instances have been switched around.

To this end, we "lift" the propositional functions from before, to better generalize to unseen environments. Given a command like "Take the block to blue room," the corresponding lifted propositional function takes the form blockInRoom block0 roomIsBlue, denoting that the block should end up in the room that is blue. We then assume an environment-specific grounding module (see Sec. 3.5.5) that consumes these lifted reward functions and performs the actual low-level binding to specific room instances, which can then be passed to a planner.

In order to effectively ground commands across multiple levels of complexity, we assume a predefined hierarchy over the state-action space of the given

grounding environment. Furthermore, each level of this hierarchy requires its own set of reward functions for all relevant tasks and sub-tasks. In our work, fast planning and the ability to ground and solve individual subtasks without needing to solve the entire planning problem make AMDPs a reliable choice for the hierarchical planner (Gopalan et al., 2017). Finally, we assume that all commands are generated from a single, fixed level of abstraction.

Given a natural language command c, we find the corresponding level of the abstraction hierarchy l, and the lifted reward function m that maximizes the joint probability of l, m given c. Concretely, we seek the level of the state-action hierarchy $\hat{l}$ and the lifted reward function $\hat{m}$ such that:

$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(l, m \mid c) \tag{2.1}$$

For example, as illustrated in Fig. 2.1, a high-level natural language command like "Take the block to the blue room" would map to the highest abstraction level, while a low-level command like "Go north a little bit" would map to the finest-grained level. We estimate this joint probability by learning a language model (described in Sec. 2.5) and training on a parallel corpus that pairs natural language commands with a corresponding reward function at a particular level of the abstraction hierarchy.

Given this parallel corpus, we train each model by directly maximizing the joint probability from Eqn. 2.1. Specifically, we learn parameters $\hat{\theta}$ that maximize the corpus likelihood:

$$\hat{\theta} = \arg\max_{\theta} \prod_{(c,l,m)\in\mathbb{C}} Pr(l, m \mid c, \theta) \tag{2.2}$$

At inference time, given a language command c, we find the best l, m that maximize the probability $Pr(l, m \mid c, \hat{\theta})$. The lifted reward function m is then completed by the grounding module (see Sec. 3.5.5) and passed to a hierarchical planner, which plans the corresponding task at abstraction level l.

2.5 Language Models

We compare four language models: an IBM Model 2 translation model (similar to MacGlashan et al. (2015)), a deep neural network bag-of-words language

model, and two recurrent neural network (RNN) language models, with varying architectures. For detailed descriptions and implementations of all the presented models, as well as the datasets used throughout this paper, please refer to the supplemental repository: https://github.com/h2r/GLAMDP.

2.5.1 IBM Model 2

As a baseline, task grounding is formulated as a machine translation problem, with natural language as the source language and semantic task representations (lifted reward functions) as the target language. We use the well-known IBM Model 2 (IBM2) machine translation model Brown et al., 1990; Brown et al., 1993 as a statistical language model for scoring reward functions given input commands.

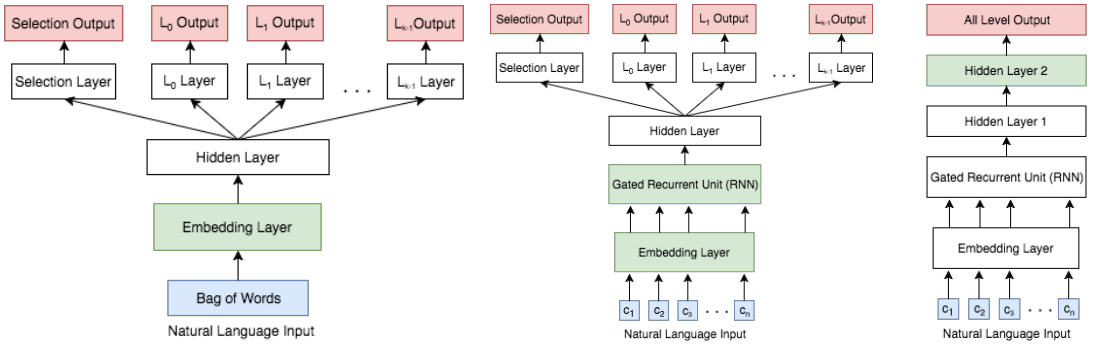
IBM2 is a generative model that solves the following objective (equivalent to Eqn. 2.1 by Bayes' rule):

$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(l, m) \cdot Pr(c \mid l, m) \tag{2.3}$$

This task grounding formulation follows directly from MacGlashan et al. (2015) and we continue in an identical fashion training the IBM2 using the standard EM algorithm.

We take a standard approach to training our IBM2 with a "bake-in" period where the EM algorithm is run for a set number of iterations only for translation parameter (τ) updates. We then learn follow with regular iterations of the EM algorithm where both the translation parameters (τ) and the alignment parameters (δ) are updated. We estimate the length parameters (η) using Maximum-Likelihood estimation.

At inference time, to pick the (l, m) tuple that maximizes the objective from Equation 2.3 we calculate the IBM2 probability for every possible (l, m) combination, using the IBM2 as a re-ranker over the possible reward function translations. We find this gives significantly better results than beam-search decoding due to the relatively small size of the reward function space, as well the formulaic nature of each reward function string.



(A) Multi-NN (B) Multi-RNN (C) Single-RNN

FIGURE 2.2: Model architectures for all three sets of deep neural network models. In blue are the network inputs, and in red are the network outputs. Going left to right, the green denotes significant structural differences between models.

2.5.2 Neural Network Language Models

We develop three classes of neural network architectures (see Fig. 2.2): a feed-forward network that takes a natural language command encoded as a bag-of-words and has separate parameters for each level of abstraction (**Multi-NN**), a recurrent network that takes into account the order of words in the sequence, also with separate parameters (**Multi-RNN**), and a recurrent network that takes into account the order of words in the sequence and has a shared parameter space across levels of abstraction (**Single-RNN**).

Multi-NN: Multiple Output Feed-Forward Network

We propose a feed-forward neural network (Bengio et al., 2000; Iyyer et al., 2015; Mikolov et al., 2013) that takes in a natural language command c as a bag-of-words vector $\vec{c}$, and outputs both the probability of each of the different levels of abstraction, as well as the probability of each reward function. We decompose the conditional probability from Eqn. 2.1 as $Pr(l, m \mid c) = Pr(l \mid c) \cdot Pr(m \mid l, c)$. Applying this to the corpus likelihood (Eqn. 2.2) and taking logarithms, the Multi-NN objective is to find parameters $\hat{\theta}$:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(\vec{c},l,m)} \log Pr(l \mid \vec{c}, \theta) + \log Pr(m \mid l, \vec{c}, \theta) \tag{2.4}$$

To learn this set of parameters, we use the architecture shown in Fig. 2.2a. Namely, we employ a multi-output deep neural network with an initial embedding layer,

a hidden layer that is shared between each of the different outputs, and then output-specific hidden and read-out layers, respectively.

The level selection output is a k-element discrete distribution, where k is the number of levels of abstraction in the given planning hierarchy. Similarly, the reward function output at each level L_i is an r_i-element distribution, where r_i is the number of reward functions at the given level of the hierarchy.

To train the model, we minimize the sum of the cross-entropy loss on each term in Eqn. 2.4. We train the network via backpropagation, using the Adam Optimizer (Kingma and Ba, 2014), with a mini-batch size of 16, and a learning rate of 0.001. Furthermore, to better regularize the model and encourage robustness, we use Dropout (Srivastava et al., 2014) after the initial embedding layer, as well as after the output-specific hidden layers with probability $p = 0.5$.

Multi-RNN: Multiple Output Recurrent Network

Inspired by the success of recurrent neural networks (RNNs) in NLP tasks (Cho et al., 2014; Mikolov et al., 2010; Mikolov et al., 2011; Sutskever, Vinyals, and Le, 2014), we propose an RNN language model that takes in a command as a sequence of words and, like the Multi-NN bag-of-words model, outputs both the probability of each of the different levels of abstraction, as well as the probability of each reward function, at each level of abstraction. RNNs extend feed-forward networks to handle variable length inputs by employing a set of one or more hidden states, which are updated after reading in each input token. Instead of converting natural language command c to a vector $\vec{c}$, we use an RNN to interpret it as a sequence of words $s = \langle c_1, c_2 \dots c_n \rangle$.

The Multi-RNN objective is then:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(c,l,m)} \log Pr(l \mid s, \theta) + \log Pr(m \mid l, s, \theta) \tag{2.5}$$

This modification is reflected in Fig. 2.2b, which is similar to the Multi-NN architecture, except in the lower layers where we use an RNN encoder that takes the sequence of raw input tokens and maps them into a fixed-size state vector. We use the gated recurrent unit (GRU) of Cho et al. (2014), a particular type of RNN cell that have been shown to work well on natural language sequence modeling tasks Chung et al., 2014.

Similar to the Multi-NN, we train the model by minimizing the sum of the cross-entropy loss of each of the two terms in Eqn. 2.5, with the same optimizer setup

as the Multi-NN model. Dropout is used to regularize the network after the initial embedding layer and the output-specific hidden layers.

Single-RNN: Single Output Recurrent Network

Both Multi-NN and Multi-RNN decompose the conditional probability of both the level of abstraction l and the lifted reward function m given the natural language command c as $Pr(l, m \mid c) = Pr(l \mid c) \cdot Pr(m \mid l, c)$, allowing for the explicit calculation of the probability of each level of abstraction given the natural language command. As a result, both Multi-NN and Multi-RNN create separate sets of parameters for each of the separate outputs, *i.e.* separate parameters for each level of abstraction in the underlying hierarchical planner.

Alternatively, we can directly estimate the joint probability $Pr(l, m \mid c)$. To do so, we propose a different type of RNN model that takes in a natural language command as a sequence of words s (as in Multi-RNN), and directly outputs the joint probability of each tuple (l, m), where l denotes the level of abstraction, and m denotes the lifted reward function at the given level.

The Single-RNN objective is to find $\hat{\theta}$ such that:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(n,l,m)} \log Pr(l, m \mid s, \theta) \tag{2.6}$$

With this Single-RNN model, we are able to significantly improve model efficiency compared to the Multi-RNN model, as all levels of abstraction share a single set of parameters. Furthermore, removing the explicit calculation of the level selection probabilities allows for the possibility of positive information transfer between levels of abstraction, which is not necessarily possible with the previous models.

The Single-RNN architecture is shown in Fig. 2.2c. We use a single-output RNN, similar to the Multi-RNN architecture, with the key difference being that there is only a *single* output, with each element of the final output vector corresponding to the probability of each tuple of levels of abstraction and reward functions (l, m) given the natural language command c.

To train the model, we minimize the cross-entropy loss of the joint probability term in Eqn. 2.6. Training hyperparameters are identical to Multi-RNN, and dropout is applied to the initial embedding layer and the penultimate hidden layer.

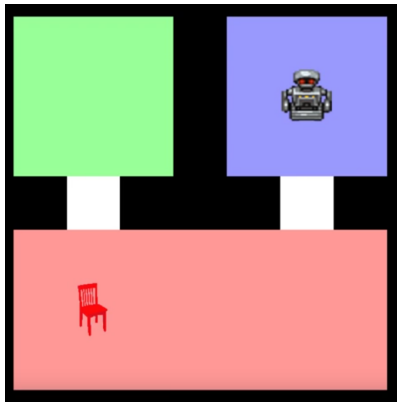


FIGURE 2.3: A starting instance of the Cleanup World Domain.

2.5.3 Grounding Module

In all of our models, the inferred lifted reward function template must be binded to environment-specific variables. The grounding module maps the lifted reward function to a grounded one that can be passed to an MDP planner. In our evaluation domain (see Fig. 2.1), it is sufficient for our grounding module to be a lookup table that maps specific environment constraints to object ID tokens. In domains with ambiguous constraints (*e.g.* a "chair" argument where multiple chairs exist), a more complex grounding module could be substituted. For instance, Artzi and Zettlemoyer (2013) present a model for executing lambda-calculus expressions generated by a combinatory categorial grammar (CCG) semantic parser, which grounds ambiguous predicates and nested arguments.

2.6 Evaluation

Our evaluation tests the hypothesis that hierarchical structure improves the speed and accuracy of language grounding at multiple levels of abstraction. We measure grounding accuracy and planning speed in simulation with a corpus-based evaluation, and demonstrate our system on a Turtlebot robot.

2.6.1 Mobile-Manipulation Robot Domain

The Cleanup World domain (Junghanns and Schaeeer, 1997; MacGlashan et al., 2015), illustrated in Fig. 2.3, is a mobile-manipulator robot domain that is partitioned into rooms (denoted by unique colors) with open doors. Each room may contain some number of objects which can be moved (pushed) by the robot. This problem is modeled after a mobile robot that moves objects around, analogous to a robotic forklift operating in a warehouse or a pick-and-place robot

Level	Example Command	Reward Function
L_0	Turn and move one spot to the right.	goWest
	Go three down, four over, two up.	agentInRoom agent0 roomIsGreen
L_1	Go to door, enter red room, push chair to green room door.	blockInRegion block0 roomIsGreen
	Go to the door then go into the red room.	agentInRegion agent0 roomIsRed
L_2	Go to the green room.	agentInRegion agent0 roomIsGreen
	Bring the chair to the blue room.	blockInRegion block0 roomIsBlue

TABLE 2.1: Example commands and corresponding reward functions.

in a home environment. We use an AMDP from Gopalan et al. (2017) for the Cleanup World domain, which imposes a three-level abstraction hierarchy for planning.

The combinatorially large state space of Cleanup World simulates real-world complexity and is ideal for exploiting abstractions. At the lowest level of abstraction L_0, the (primitive) action set available to the robot agent consists of north, south, east, and west actions. Users directing the robot at this level of granularity must specify lengthy step-by-step instructions for the robot to execute. At the next level of abstraction L_1, the state space of Cleanup World only consists of rooms and doors. The robot's position is solely defined by the region (*i.e.* room or door) it resides in. Abstracted actions are *subroutines* for moving either the robot or a specific block to a room or door. It is impossible to transition between rooms without first transitioning through a door, and it is only possible to transition between adjacent regions; any language guiding the robot at L_1 must adhere to these dynamics. Finally, the highest level of abstraction, L_2, removes the concept of doors, leaving only rooms as regions; all L_1 transition dynamics still hold, including adjacency constraints. Subroutines exist for moving either the robot or a block between connected rooms. The full space of subroutines at all levels and their corresponding propositional functions are defined by (Gopalan et al., 2017). Fig. 2.1 shows a few collected sample commands at each level and the corresponding level-specific AMDP reward function.

2.6.2 Procedure

We conducted an Amazon Mechanical Turk (AMT) user study to collect natural language samples at various levels of abstraction in Cleanup World. Annotators were shown video demonstrations of ten tasks, always starting from the state shown in Fig. 2.3. For each task, users provided a command that they would give to a robot, to perform the action they saw in the video, while constraining their language to adhere to one of three possible levels in a designated abstraction hierarchy: fine-grained, medium, and coarse. This data provided multiple

	Evaluated L_0	Evaluated L_1	Evaluated L_2
Trained L_0	**21.61%**	**17.20%**	21.87%
Trained L_1	9.83%	10.23%	13.90%
Trained L_2	14.94%	12.84%	**31.49%**

FIGURE 2.4: IBM2 Reward Grounding Baselines

parallel corpora for the machine translation problem of task grounding. We measured our system's performance by passing each command to the language grounding system and assessing whether it inferred both the correct level of abstraction and the reward function. We also recorded the response time of the system, measuring from when the command was issued to the language model to when the (simulated) robot would have started moving. Accuracy values were computed using the mean of multiple trials of ten-fold cross validation. The space of possible tasks included moving a single step as well as navigating to a particular room, taking a particular object to a designated room, and all combinations thereof.

Unlike MacGlashan et al. (2015), the demonstrations shown were not only limited to simple robot navigation and object placement tasks, but also included composite tasks (*e.g.* "Go to the red room, take the red chair to the green room, go back to the red room, and return to the blue room"). Commands reflecting a clear misunderstanding of the presented task, *e.g.* "please robot", were removed from the dataset. Such removals were rare; we removed fewer than 30 commands for this reason, giving a total of 3047 commands. Per level, there were 1309 L_0 commands, 872 L_1 commands, and 866 L_2 commands. The L_0 corpus included more commands since the tasks of moving the robot one unit in each of the four cardinal directions do not translate to higher levels of abstraction.

2.6.3 Robot Task Grounding

We present the baseline task grounding accuracies in Fig. 2.4 and 2.5 to demonstrate the importance of inferring the latent abstraction level in language. We simulate the effect of an oracle that partitions all of the collected AMT commands into separate corpora according to the specificity of each command. For this experiment, any L_0 commands that did not exist at all levels of the Cleanup World hierarchy were omitted, resulting in a condensed L_0 dataset of 869 commands. We trained multiple IBM2 and Single-RNN models using data from one distinct level and then evaluated using data from a separate level. Training a model at a particular level of abstraction includes grounding solely to the reward functions that exist at that same level. Reward functions at the evaluation level were mapped to the equivalent reward functions at the training level (*e.g.*

	Evaluated L_0	Evaluated L_1	Evaluated L_2
Trained L_0	**77.67%**	28.05%	23.26%
Trained L_1	32.79%	**82.99%**	74.65%
Trained L_2	14.19%	58.62%	**87.91%**

FIGURE 2.5: Single-RNN Reward Grounding Baselines

L_1 agentInRegion to L_0 agentInRoom). Entries along the diagonal represent the average task grounding accuracy for multiple, random 90-10 splits of the data at the given level. Otherwise, evaluation checked for the correct grounding of the command to a reward function at the training level equivalent to the true reward function at the alternate evaluation level.

Task grounding scores are uniformly quite poor for IBM2; however, IBM2 models trained using L_0 and L_2 data respectively result in models that substantially outperform those trained on alternate levels of data. It is also apparent that an IBM2 model trained on L_1 data fails to identify the features of the level. We speculate that this is caused, in part, by high variance among the language commands collected at L_1 as well as the large number of overlapping, repetitive tokens that are needed for generating a valid machine language instance at L_1. While these models are worse than what MacGlashan et al. (2015) observed, we note that we do not utilize a task or behavior model. It follows that integrating one or both of these components would only help prune the task grounding space of highly improbable tasks and improve our performance.

Conversely, Single-RNN shows the expected maximization along diagonal entries that comes from training and evaluating on data at the same level of abstraction. These show that a model limited to a single level of language abstraction is not flexible enough to deal with the full scope of possible commands. Additionally, Single-RNN demonstrates more robust task grounding than statistical machine translation.

The task grounding and level inference scores for the models in Sec. 2.5 are shown in Fig. 2.6. Attempting to embed the latent abstraction level within the machine language of IBM2 results in weak level inference. Furthermore, grounding accuracy falls even further due to sparse alignments and the sharing of tokens between tasks in machine language (*e.g.* agentInRoom agent0 room1 at L_0 and agentInRegion agent0 room1 at L_1). The fastest of all the neural models, and the one with the fewest number of parameters overall, Multi-NN shows notable improvement in level inference over the IBM2; however, task grounding performance still suffers, as the bag-of-words representation fails to capture the sequential word dependencies critical to the intent of each command. Multi-RNN again improves upon level prediction accuracy and leverages the high-dimensional representation learned by initial RNN layer to train reliable

	Level Selection	Reward Grounding
IBM2	79.87%	27.26%
Multi-NN	93.51%	36.05%
Multi-RNN	95.71%	80.11%
Single-RNN	**95.91**%	**80.46**%

FIGURE 2.6: Accuracy of 10-Fold Cross Validation (averaged over 3 runs) for each of the models on the AMT Dataset.

grounding models specific to each level of abstraction.

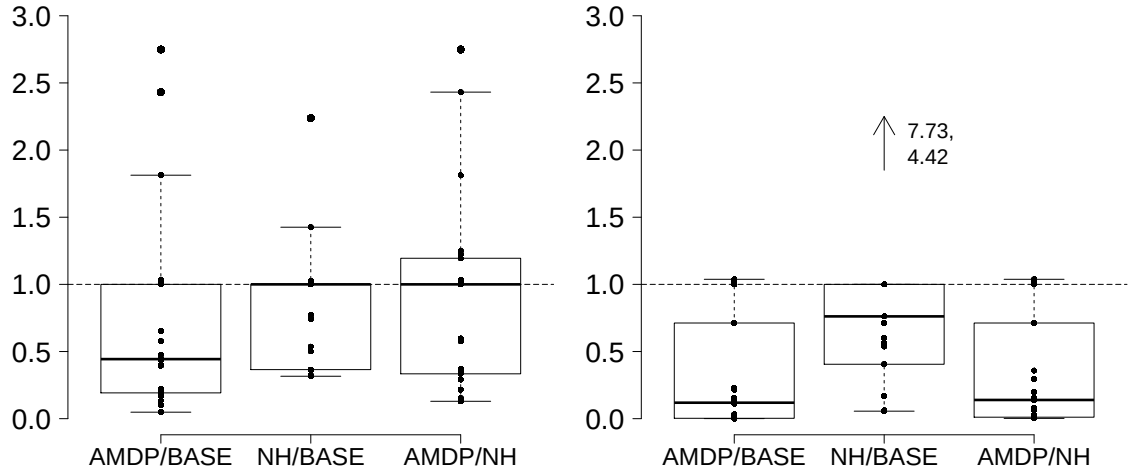
Finally, Single-RNN has near-perfect level prediction and demonstrates the successful learning of abstraction level as a latent feature within the neural model. By not using an oracle for level inference, there is a slight loss in performance compared to the results obtained in Fig. 2.5; however, we still see improved grounding performance over Multi-RNN that can be attributed to the full sharing of parameters across all training samples allowing for positive information transfer between abstraction levels.

2.6.4 Robot Response Time

Fast response times are important for fluid human-robot interaction, so we assessed the time it would take a robot to respond to natural language commands in our corpus. We measured the time it takes for the system to process a natural language command, map it to a reward function, and then solve the resulting MDP to yield a policy so that the simulated robot would start moving. We used Single-RNN for inference since it was the most accurate grounding model, and only correctly grounded instances were evaluated, so our results are for 2634 of 3047 commands that Single-RNN got correct.

We compared three different planners to solve the MDP:

- **BASE**: A state-of-the-art flat (non-hierarchical) planner, bounded real-time dynamic programming (BRTDP (McMahan, Likhachev, and Gordon, 2005)).
- **AMDP**: A hierarchical planner for MDPs (Gopalan et al., 2017). At the primitive level of the hierarchy (L_0), **AMDP** also requires a flat planner; we use **BASE** to allow for comparable planning times. Because the subtasks have no compositional structure, a Manhattan-distance heuristic can be used at L_0. While **BASE** technically allows for heuristics, distance-based heuristics are unsuitable for the composite tasks in our dataset. This illustrates another benefit of using hierarchies: to decompose composite tasks into subtasks that are amenable to better heuristics.



(A) Regular domain (2^{14} states) (B) Large domain (2^{18} states)

FIGURE 2.7: Relative inference + planning times for different planning approaches on the same correctly grounded AMT commands. For each method pair, values less than 1 indicate the method on the numerator (left of '/') is better. Each data point is an average of 1000 planning trials.

- **NH** (No Heuristic): Identical to **AMDP**, but without the heuristic as a fair comparison against **BASE**.

We hypothesize **NH** is faster than **BASE** (due to use of hierarchy), but not as fast as **AMDP** (due to lack of heuristics).

Since the actual planning times depend heavily on the actual task being grounded (ranging from 5ms for goNorth to 180s for some high-level commands), we instead evaluate the *relative* times used between different planning approaches. Fig. 2.7a shows the results for all 3 pairs of planners. For example, the left-most column shows $\frac{\textbf{AMDP time}}{\textbf{BASE time}}$; the fact that most results were less than 1 indicates that **AMDP** usually outperforms **BASE**. Using Wilcoxon signed-rank tests, we find that each approach in the numerator is significantly faster ($p < 10^{-40}$) than the one in the denominator, *i.e.* **AMDP** is faster than **NH**, which is in turn faster than **BASE**; this is consistent with our hypothesis. Comparing **AMDP** to **BASE**, we find that **AMDP** is twice as fast in over half the cases, 4 times as fast in a quarter of the cases, and can reach 20 times speedup. However, **AMDP** is also slower than **BASE** on 23% of the cases; of these, half are within 5% of **BASE**, but the other half is up to 3 times slower. Inspecting these cases suggests that the slowdown is due to overhead from instantiating multiple planning tasks in the hierarchy; this overhead is especially prominent in relatively small domains like Cleanup World. Note that in the worst case this is less than a 2s absolute time difference.

From a computational standpoint, the primary advantage of hierarchy is space and time abstraction. To illustrate the potential benefit of using hierarchical planners in larger domains, we doubled the size of the original Cleanup domain and ran the same experiments. Ideally, this should have no effect on L_1 and L_2 tasks, since these tasks are agnostic to the discretization of the world. The results are shown in Fig. 2.7b, which again are consistent with our hypothesis. Somewhat surprisingly though, while **NH** still outperforms **BASE** ($p < 10^{-150}$), it was much less efficient than **AMDP**, which shows that the hierarchy itself was insufficient; the heuristic also plays an important role. Additionally, **NH** suffered from two outliers, where the planning problem became more complex because the solution was constrained to conform to the hierarchy; this is a well-known tradeoff in hierarchical planning (Dieterrich, 2000). The use of heuristics in **AMDP** mitigated this issue. **AMDP** times almost stayed the same compared to the regular domain, hence outperforming **BASE** and **NH** ($p < 10^{-200}$). The larger domain size also reduced the effect of hierarchical planning overhead: **AMDP** was only slower than **BASE** in 10% of the cases, all within $< 4\%$ of the time it took for **BASE**. Comparing **AMDP** to **BASE**, we find that **AMDP** is 8 times as fast in over half the cases, 100 times as fast in a quarter of the cases, and can reach up to 3 orders of magnitude in speedup. In absolute time, **AMDP** took $<$ 1s on 90% of the tasks; in contrast, **BASE** takes $>$ 20s on half the tasks.

2.6.5 Robot Demonstration

Using the trained grounding model and the corresponding AMDP hierarchy, we tested with a Turtlebot on a small-scale version of the Cleanup World domain. To accommodate the continuous action space of the Turtlebot, the low-level, primitive actions at L_0 of the AMDP were swapped out for move forward, backward, and bidirectional rotation actions; all other levels of the AMDP remained unchanged. The low level commands used closed loop control policies, which were sent to the robot using the Robot Operating System Quigley et al., 2009.

Spoken commands were provided by an expert human user instructing the robot to navigate from one room to another. These verbal commands were converted from speech to text using Google's Speech API (*Google Speech API* 2017) before being grounded with the trained Single-RNN model. The resulting grounding, with both the AMDP hierarchy level and reward function, fed directly into the AMDP planner resulting in almost-instantaneous planning and execution. Numerous commands ranging from the low-level "Go north" all the way to the high-level "Take the block to the green room" were planned and executed using the AMDP with imperceivable delays after the conversion from speech to text. A video demonstration of the end-to-end system is available online: https://youtu.be/9bU2oE5RtvU

2.7 Discussion

Overall, our best grounding model, Single-RNN, performed very well, correctly grounding commands much of the time; however, it still experienced errors. At the lowest level of abstraction, the model experienced some confusion between robot navigation (agentInRoom) and object manipulation (blockInRoom) tasks. In the dataset, some users explicitly mention the desired object in object manipulation tasks while others did not; without explicit mention of the object, these commands were almost identical to those instructing the robot to navigate to a particular room. For example, one command that was correctly identified as instructing the robot to take the chair to the green room in Fig. 2.3 is "Go down...west until you hit the chair, push chair north..." A misclassified command for the same task was "Go south...west...north..." These commands ask for the same directions with the same amount of repetition (omitted) but only one mentions the object of interest allowing for the correct grounding. Overall, 83.3% of green room navigation tasks were grounded correctly while 16.7% were mistaken for green room object manipulation tasks.

Another source of error involved an interpretation issue in the video demonstrations presented to users. The robot agent shown to users as in Fig. 2.3 faces south and this orientation was assumed by the majority of users; however, some users referred to this direction as north (in the perspective of the robot agent). This confusion led to some errors in the grounding of commands instructing the robot to move a single step in one of the four cardinal directions. Logically, these conflicts in language caused errors for each of the cardinal directions as 31.25% of north commands were classified as south and 15% of east commands were labeled as west.

Finally, there were various forms of human error throughout the collected data. In many cases, users committed typos that actually affected the grounding result (*e.g.* asking the robot to take the chair back to the green room instead of the observed blue room). For some tasks, users often demonstrated some difficulty understanding the abstraction hierarchy described to them resulting in commands that partially belong to a different level of abstraction than what was requested. In order to avoid embedding a strong prior or limiting the natural variation of the data, no preprocessing was performed in an attempt to correct or remove these commands. A stronger data collection approach might involve adding a human validation step and asking separate users to verify that the supplied commands do translate back to the original video demonstrations under the given language constraints as in MacMahon, Stankiewicz, and Kuipers (2006).

2.8 Conclusion

We presented a system for interpreting and grounding natural language commands to a mobile-manipulator robot at multiple levels of abstraction. To our knowledge, our system is not only the first work to ground language at multiple levels of abstraction, but also the first to use deep neural networks for language grounding on robots. Our proposed language-grounding models significantly outperform the previous state-of-the-art method for mapping natural language commands to reward functions. By explicitly considering the level of abstraction, our system can interpret a much wider range of natural language commands, as well as leverage an existing hierarchical planner for efficient planning and execution of robot tasks. Finally, our Turtlebot evaluation demonstrates that this system works well in real-world environments and is an encouraging step towards seamless human-robot interaction.

To achieve natural interaction with humans, robots must be able to interpret all possible natural language input. Therefore, we must weaken the constraints and assumptions we place on input from users. To this end, we plan to extend our proposed models to handle natural language commands specified at a *mixture* of abstraction levels. More generally, we should not allow an existing planning abstraction hierarchy to constrain our interpretation of language. In contrast, we can use the space of user inputs to *inform* the learning of appropriate abstraction hierarchies, aiming to find structures that both match user language and are efficient to plan with.

We envision that our system is applicable to a large variety of real-world scenarios, particularly in environments where multiple levels of abstraction naturally occur, such as in surgical, manufacturing, and household robotics.

Chapter 3

Grounding Actions and Goals

This chapter consists of work that was done jointly with Eddie Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex[1], presented as a workshop paper at Language Grounding for Robotics (RoboNLP), at Association for Computational Linguistics 2017.

The central theme of this chapter is **generalization**. We present methods for grounding language commands that either specify actions or goals to a modular representation; specifically, we show how this representation lends itself well to generalization to unseen tasks at test time.

3.1 Abstract

Robots operating alongside humans in diverse, stochastic environments must be able to accurately interpret natural language commands. These instructions often fall into one of two categories: those that specify a goal condition or target state, and those that specify explicit actions, or how to perform a given task. Recent approaches have used reward functions as a semantic representation of goal-based commands, which allows for the use of a state-of-the-art planner to find a policy for the given task. However, these reward functions cannot be directly used to represent action-oriented commands. We introduce a new hybrid approach, the Deep Recurrent Action-Goal Grounding Network (DRAGGN), for task grounding and execution that handles natural language from either category as input, and generalizes to unseen environments. Our robot-simulation results demonstrate that a system successfully interpreting both goal-oriented and action-oriented task specifications brings us closer to robust natural language understanding for human-robot interaction.

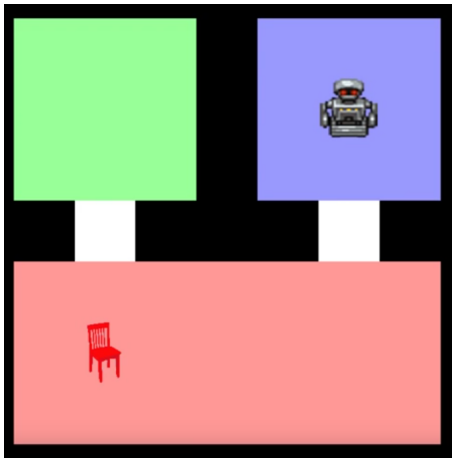


FIGURE 3.1: Sample configuration of the Cleanup World mobile-manipulator domain (MacGlashan et al., 2015), used throughout this work. A possible goal-based instruction could be "Take the chair to the green room," while a possible action-based instruction could be "Go three steps south, then two steps west."

3.2 Introduction

Natural language affords a convenient choice for delivering instructions to robots, as it offers flexibility, familiarity, and does not require users to have knowledge of low-level programming. In the context of grounding natural language instructions to tasks, human-robot instructions can be interpreted as either high-level goal specifications or low-level instructions for the robot to execute.

Goal-oriented commands define a particular target state specifying where a robot should end up, whereas action-oriented commands specify a particular sequence of actions to be executed. For example, a human instructing a robot to "go to the kitchen" outlines a goal condition to check if the robot is in the kitchen. Alternatively, a human providing the command "take three steps to the left" defines a trajectory for the robot to execute. We need to consider both forms of commands to understand the full space of natural language that humans may use to communicate their intent to robots. While humans also combine commands of both types into a single instruction, we make the simplifying assumption that a command belongs entirely to a single type and leave the task of handling mixtures and compositions to future work.

Existing approaches can be broadly divided into one of two regimes. Goal-based approaches like MacGlashan et al. (2015) and Arumugam et al. (2017) leverage some intermediate task representation and then automatically find a low-level

[1]Siddharth Karamcheti, Edward Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson L. S. Wong, Stefanie Tellex: "A Tale of Two DRAGGNs: A Hybrid Approach for Interpreting Action-Oriented and Goal-Oriented Instructions". *Workshop in Language Grounding for Robotics (RoboNLP) at Association for Computational Linguistics 2017*

trajectory to achieve the goal using a planner. Other approaches, in the action-oriented regime, directly infer action sequences (Tellex et al., 2011; Matuszek et al., 2012; Artzi and Zettlemoyer, 2013; Andreas and Klein, 2015) from the syntactic or semantic parse structure of natural language. However, these approaches can be computationally intractable for large state-action spaces or use ad-hoc methods to execute high-level language rather than relying on a planner. Furthermore, these methods are unable to adapt to dynamic changes in the environment; for example, consider an environment in which the wind, or some other force moves an object that a robot has been tasked with picking. Action sequence based approaches would fail to handle this without additional user input, while goal-based approaches would be able to re-plan on the fly, and complete the task.

To address the issue of dealing with both goal-oriented and action-oriented commands, we present a new language grounding framework that, given a natural language command, is capable of inferring the latent command type. Recent approaches leveraging deep neural networks have formulated the language grounding problem as sequence-to-sequence learning or multi-label classification (Mei, Bansal, and Walter, 2016; Arumugam et al., 2017). Inspired by the recent success of neural networks to model programs that are highly compositional and sequential in nature, we present the Deep Recurrent Action/Goal Grounding Network (DRAGGN) framework, derived from the the Neural Programmer-Interpreter (NPI) of Reed and Freitas (2016) and outlined in Section 3.5.2. We introduce two instances of DRAGGN models, each with slightly different architectures. The first, the Joint-DRAGGN (J-DRAGGN) is defined in Section 3.5.3, while the second, the Independent-DRAGGN (I-DRAGGN) is defined in Section 3.5.4.

3.3 Related Work

There has been a broad and diverse set of work examining how best to interpret and execute natural language instructions on a robot platform (Vogel and Jurafsky, 2010; Tellex et al., 2011; Artzi and Zettlemoyer, 2013; Howard, Tellex, and Roy, 2014; Andreas and Klein, 2015; Hemachandra et al., 2015; MacGlashan et al., 2015; Paul et al., 2016; Mei, Bansal, and Walter, 2016; Arumugam et al., 2017). Vogel and Jurafsky (2010) produce policies using language and expert trajectories based rewards, which allow for planning within a stochastic environment along with re-planning in case of failure. (Tellex et al., 2011) instead grounds language to trajectories satisfying the language specification. Howard, Tellex, and Roy, 2014 chose to ground language to constraints given to an external planner, which is a much smaller space to perform inference over than trajectories. MacGlashan et al. (2015) formulate language grounding as a machine translation problem, treating propositional logic functions as both a machine language

and reward function. Reward functions or cost functions can allow richer descriptions of trajectories than plain constraints, as they can describe preferential paths. Additionally, Arumugam et al. (2017) simplify the problem from one of machine translation to multi-class classification, learning a deep neural network to map arbitrary natural language instructions to the corresponding reward function.

Informing our distinction between action sequences and goal state representation is the division presented by Dzifcak et al. (2009b), who posited that natural language can be interpreted as *both* a goal state specification and an action specification. Rather than producing both from each language command, our DRAGGN framework makes the simplifying assumption that only one representation captures the semantics of the language; additionally, our framework does not require a manually pre-specified grammar.

Recently, deep neural networks have found widespread success and application to a wide array of problems dealing with natural language (Bengio et al., 2000; Mikolov et al., 2010; Mikolov et al., 2011; Cho et al., 2014; Chung et al., 2014; Iyyer et al., 2015). Unsurprisingly, there have been some initial steps taken towards applying neural networks to language grounding problems. Mei, Bansal, and Walter (2016) uses a recurrent neural network (RNN) with long short-term memory (LSTM) cells (Hochreiter and Schmidhuber, 1997) to learn sequence-to-sequence mappings between natural language and robot actions. This model augments the standard sequence-to-sequence architecture by learning parameters that represent latent alignments between natural language tokens and robot actions. Arumugam et al. (2017) used an RNN-based model to produce grounded reward functions at multiple levels of an Abstract Markov Decision Process hierarchy (Gopalan et al., 2016), varying the abstraction level with the level of abstraction used in natural language.

Our DRAGGN framework is related to the Neural Programmer-Interpreter (NPI) (Reed and Freitas, 2016). The original NPI model is a controller trained via supervised learning to interpret and learn when to call specific programs (or subprograms), which arguments to pass into the currently active program, and when to terminate execution of the current program. We draw a parallel between inferred NPI programs and our method of predicting either lifted reward functions or action trajectories.

3.4 Problem Setting

We consider the problem of mapping from natural language to robot actions within the context of Markov decision processes. A Markov decision process (MDP) is a five-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ defining a state space $\mathcal{S}$, action space $\mathcal{A}$,

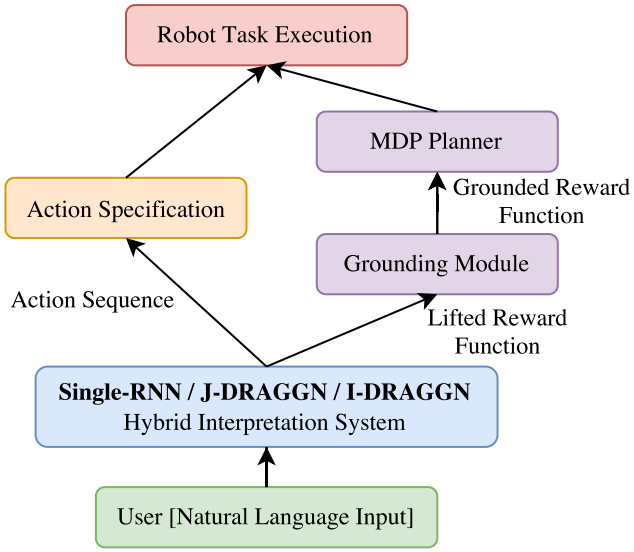


FIGURE 3.2: System for grounding both action-oriented (left branch) and goal-oriented (right branch) natural language instructions to executable robot tasks. Our main contribution is the hybrid interpretation system (blue box), for which we present two novel models based on the DRAGGN framework (J-DRAGGN and I-DRAGGN) in Section 3.5.

state transition probabilities $\mathcal{T}$, reward function $\mathcal{R}$, and discount factor γ (Bellman, 1957a; Puterman, 1994). An MDP solver produces a policy that maps from states to actions in order to maximize the total expected discounted reward.

While reward functions are flexible and expressive enough for a wide variety of task specifications, they are a brittle choice for specifying an exact sequence of actions, as enumerating every possible action sequence as a reward function (i.e. a specific reward function for the sequence Up 3, Down 2) can quickly become intractable. This paper introduces models that can produce desired behavior by inferring either reward functions or primitive actions. We assume that all available actions $\mathcal{A}$ and the full space of potential reward functions (*i.e.*, the full space of possible tasks) are known *a priori*. When a reward function is predicted by the model, an MDP planner is applied to derive the resultant policy (see system pipeline Figure 3.2).

We focus our evaluation on the the Cleanup World mobile-manipulator domain (MacGlashan et al., 2015; Arumugam et al., 2017). The Cleanup World domain consists of an agent in a 2-D world with uniquely colored rooms and movable objects. A domain instance is shown in Figure 3.1. The domain itself is implemented as an object-oriented Markov decision process (OO-MDP) where states are denoted entirely by collections of objects, with each object having its own identifier, type, and set of attributes (Diuk, Cohen, and Littman, 2008). Domain objects include rooms and interactable objects (*e.g* a chair, basket, etc.) all of which have location and color attributes. Propositional logic functions can be

used to identify relevant pieces of an OO-MDP state and their attributes; as in MacGlashan et al. (2015) and Arumugam et al. (2017), we treat these propositional functions as reward functions. In Figure 3.1, the goal-oriented command "take the chair to the green room" may be represented with the reward function `blockInRoom block0 room1`, where the `blockInRoom` propositional function checks if the location attribute of `block0` is contained in `room1`.

3.5 Approach

We now outline the pipeline that converts natural language input to robot behavior. We begin by first defining the semantic task representation used by our grounding models that comes directly from the OO-MDP propositional functions of the domain. Next, we examine our novel DRAGGN framework for language grounding and, in particular, address the separate paths taken by action-oriented and goal-oriented commands through the system as seen in Figure 3.2. Finally, we discuss two different implementations of the DRAGGN framework that make different assumptions about the relationship between tasks and constraints. Specifically, we introduce the Joint-DRAGGN (J-DRAGGN), that assumes a probabilistic dependence between tasks (i.e. `goUp`) and the corresponding arguments (i.e. `5 steps`) based on a natural language instruction, and the Independent-DRAGGN (I-DRAGGN) that treats tasks and arguments as independent given a natural language instruction.

3.5.1 Semantic Representation

In order to map arbitrary natural language instructions to either action trajectories or goal conditions, we require a compact but sufficiently expressive semantic representation for both. To this end, we define the *callable unit*, which takes the form of a single-argument function. These functions are paired with *binding arguments* whose possible values depend on the callable unit type. As in MacGlashan et al. (2015) and Arumugam et al. (2017), our approach generates reward function templates, or *lifted* reward functions, for goal-oriented tasks along with environment-specific constraints. Once these templates and constraints are resolved to get a grounded reward function, the associated goal-oriented tasks can be solved by an off-the-shelf planner thereby improving transfer and generalization capabilities.

Goal-oriented callable units (lifted reward functions) are paired with binding arguments that specify properties of environment entities that must be satisfied in order to achieve the goal. These binding arguments are later resolved by the Grounding Module (see Section 3.5.5) to produce grounded reward functions (OO-MDP propositional logic functions) that are handled by an MDP planner.

Action-Oriented	Goal-Oriented
goUp(numSteps)	agentInRoom(room)
goDown(numSteps)	blockInRoom(room)
goLeft(numSteps)	
goRight(numSteps)	

TABLE 3.1: Set of action-oriented and goal-oriented callable units that can be generated by our DRAGGN models in the Cleanup World domain.

Action-oriented callable units directly correspond to the primitive actions available to the robot and are paired with binding arguments defining the number of sequential executions of that action. The full set of callable units along with requisite binding arguments is shown in Table 3.1.

3.5.2 Deep Recurrent Action/Goal Grounding Network

While the Single-RNN model of Arumugam et al. (2017) is effective, it cannot model the compositional argument structure of language. A unit-argument pair not observed at training time will not be predicted from input data, even if the constituent pieces were observed separately. Additionally, the Single-RNN model requires every possible unit-argument pair to be enumerated, to form the output space. As the environment grows to include more objects with richer attributes, this output space becomes intractable.

To resolve this, we introduce the Deep Recurrent Action/Goal Grounding Network (DRAGGN) framework. Unlike previous approaches, the DRAGGN framework maps natural language instructions to *separate* distributions over callable units and (possibly multiple) binding constraints, generating either action sequences or goal conditions. By treating callable units and binding arguments as separate entities, we circumvent the combinatorial dependence on the size of the domain.

This unit-argument separation is inspired by the Neural Programmer-Interpreter (NPI) of Reed and Freitas (2016). The callable units output by DRAGGN are analogous to the subprograms output by NPI. Additionally, both NPI and DRAGGN allow for subprograms/callable units with an arbitrary number of arguments (by adding a corresponding number of Binding Argument Networks, as shown at the top right of Figure 3.3a, each with its own output space).

We assume that each natural language instruction can be represented by a single unit-argument pair with only one argument. Consequently, in our experiments, we assume that sentences specifying sequences of commands have been segmented, and each segment is given to the model one at a time. The limitation to a single argument only arises because of the domain's simplicity; as mentioned

above, it is straightforward to extend our models to handle extra arguments by adding extra Binding Argument Networks.

To formalize the DRAGGN objective, consider a natural language instruction l. Our goal is to find the callable unit $\hat{c}$ and binding arguments $\hat{\mathbf{a}}$ that maximize the following joint probability:

$$\hat{c}, \hat{\mathbf{a}} = \arg\max_{c,\mathbf{a}} \Pr(c, \mathbf{a} \mid l) \tag{3.1}$$

Depending on the assumptions made about the relationship between callable units c and binding arguments $\mathbf{a}$, we can decompose the above objective in two ways: preserving the dependence between the two, and learning the relationship between the units and arguments jointly, and treating the two as independent. These two decompositions result in the Joint-DRAGGN and Independent-DRAGGN models respectively.

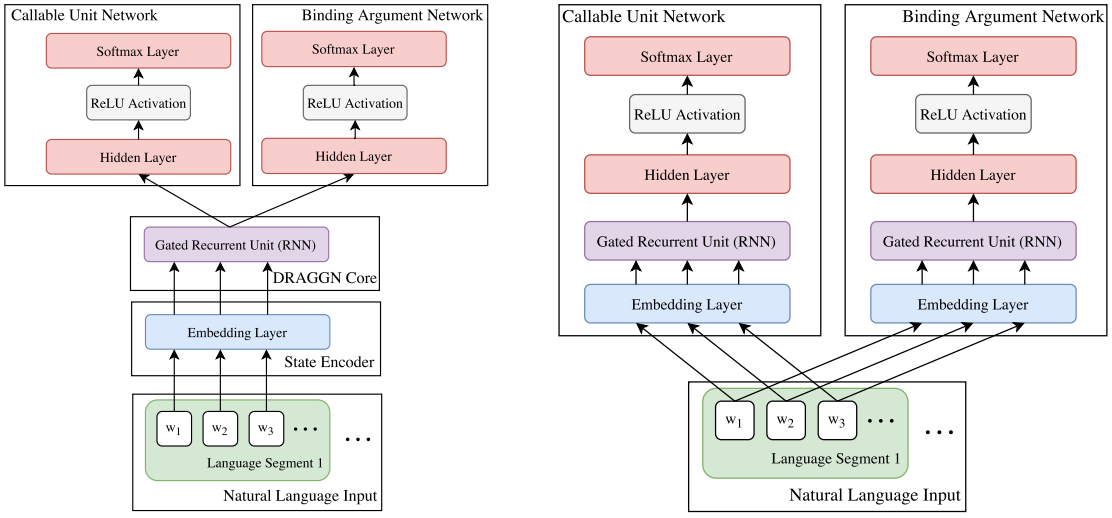
Given the training dataset of natural language and the space of unit-argument pairs, we train our DRAGGN models end-to-end by minimizing the sum of the cross-entropy losses between the predicted distributions and true labels for each separate distribution (*i.e.* over callable units and binding arguments). At inference time, we first choose the callable unit with the highest probability given the natural language instruction. We then choose the binding argument(s) with highest probability from the set of valid arguments. The validity of a binding argument given a callable unit is given *a priori*, by the specific environment, rather than being learned at training time.

Our models were trained using Adam (Kingma and Ba, 2014), for 125 epochs, with a batch size of 16, and a learning rate of 0.0001.

3.5.3 Joint DRAGGN (J-DRAGGN)

The Joint DRAGGN (J-DRAGGN) models the joint probability in Equation 3.1, coupled via the shared RNN state in the DRAGGN Core (as depicted in Figure 3.3a), but selects the optimizer sequentially, as follows:

$$\begin{aligned} \hat{c}, \hat{\mathbf{a}} &= \arg\max_{c,\mathbf{a}} \Pr(c, \mathbf{a} \mid l) \\ &\approx \arg\max_{\mathbf{a}} \left[\arg\max_{c} \Pr(c, \mathbf{a} \mid l) \right] \end{aligned} \tag{3.2}$$



(A) Joint DRAGGN (B) Independent DRAGGN

FIGURE 3.3: Architecture diagrams for the two Deep Recurrent Action/Goal Grounding Network (DRAGGN) models, introduced in Sections 3.5.3 and 3.5.4. Both architectures ground arbitrary natural language instructions to callable units (either actions or lifted reward functions), and binding arguments.

We first encode the constituent words of our natural language segment into fixed-size embedding vectors. From there, the sequence of word embeddings is fed through an RNN denoted by the DRAGGN Core[2]. After processing the entire segment, the current gated recurrent unit (GRU) hidden state is then treated as a representative vector for the entire natural language segment. This single hidden core vector is then passed to both the Callable Unit Network and the Binding Argument Network, allowing for both networks to be trained jointly, enforcing a dependence between the two.

The Callable Unit Network is a two-layer feed-forward network using rectified linear unit (ReLU) activation. It takes the DRAGGN Core output vector as input to produce a softmax probability distribution over all possible callable units. The Binding Argument Network is a separate network with an identical architecture and takes the same input, but instead produces a probability distribution over all possible binding arguments. The two models do not need to share the same architecture; for example, callable units with multiple arguments require multiple different argument networks, one for each possible binding constraint.

[2]We use the gated recurrent unit (GRU) as our RNN cell, because of its effectiveness in natural language processing tasks, such as machine translation (Cho et al., 2014), while requiring fewer parameters than the LSTM cell (Hochreiter and Schmidhuber, 1997).

3.5.4 Independent DRAGGN (I-DRAGGN)

The Independent DRAGGN (I-DRAGGN), contrary to the Joint DRAGGN, decomposes the objective from Equation 3.1 by treating callable units and binding arguments as being independent, given the original natural language instruction. More precisely, the I-DRAGGN objective is:

$$\hat{c}, \hat{\mathbf{a}} = \arg\max_{c,\mathbf{a}} \Pr(c \mid l)\ \Pr(\mathbf{a} \mid l) \tag{3.3}$$

The I-DRAGGN network architecture is shown in Figure 3.3b. Beyond the difference in objective functions, there is another key difference between the I-DRAGGN and J-DRAGGN architectures. Rather than encoding the constituent words of the natural language instruction once, and feeding the resulting embeddings through a DRAGGN Core to generate a shared core vector, the I-DRAGGN model embeds and encodes the natural language instruction *twice*, using two separate embedding matrices and GRUs, one each for the callable unit and binding argument. In this way, the I-DRAGGN model encapsulates two disjoint neural networks, each with their own individual parameter sets that are trained independently. The latter half of each individual network (the Callable Unit Network and Binding Argument Network) remains the same as that of the J-DRAGGN.

3.5.5 Grounding Module

If a goal-oriented callable unit is returned (*i.e.* a lifted reward function), we require an additional step of completing the reward function with environment-specific variables. As described in Arumugam et al. (2017), we use a Grounding Module to perform this step. The Grounding Module maps the inferred callable unit and binding argument(s) to a final grounded reward function that can be passed to an MDP planner. In our implementation, the Grounding Module is a lookup table mapping specific binding arguments to room ID tokens. A more advanced implementation of the Grounding Module would be required in order to handle domains with non-unique binding arguments (*e.g.* resolving between multiple objects with overlapping attributes).

3.6 Experiments

We assess the effectiveness of both our J-DRAGGN and I-DRAGGN models via instruction grounding accuracy for robot navigation and mobile-manipulation

Natural Language	Callable Unit	Argument
Go to the red room.	agentInRoom	roomIsRed
Put the block in the green room.	blockInRoom	roomIsGreen
Go up three spaces.	goUp	3

TABLE 3.2: Examples of natural language phrases and corresponding callable units and arguments.

tasks. As a baseline, we compare against the state-of-the-art Single-RNN model introduced by Arumugam et al. (2017).

3.6.1 Procedure

To conduct our evaluation, we use the dataset of natural language commands for the single instance of Cleanup World domain seen in Figure 3.1, from Arumugam et al. (2017). In the user study, Amazon Mechanical Turk users were presented with trajectory demonstrations of a robot completing various navigation and object manipulation tasks. Users were prompted to provide natural language commands that they believed would have generated the observed behavior. Since the original dataset was compiled for analyzing the hierarchical nature of language, we were easily able to filter the commands down to only those using high-level goal specifications and low-level trajectory specifications. This resulted in a dataset of 3734 natural language commands total.

To produce a dataset of action-specifying callable units, experts annotated low-level trajectory specifications from the Arumugam et al. (2017) dataset. For example, the command "Down three paces, then up two paces, finally left four paces" was segmented into "down three spaces," "then up two paces," "finally left four paces," and was given a corresponding execution trace of `goDown 3`, `goUp 2`, `goLeft 4`. The existing set of grounded reward functions in the dataset were converted to callable units and binding arguments. Examples of both types of language are presented in Table 3.2 with their corresponding callable unit and binding arguments.

To fully show the capabilities of our model, we tested on two separate versions of the dataset. The first is the standard dataset, consisting of a 90-10 split of the collected action-oriented and goal-oriented commands We also evaluated our models on an "unseen" dataset, which consists of a specific train-test split that evaluates how well models can predict previously unseen action sequence combinations. For example, in this dataset the training data might consist only of action sequences of the form `goUp 3`, and `goDown 4`, while the test data would only consist of the "unseen" action sequence `goUp 4`.

	Actions	Goals	Actions (Unseen)	Overall
Single-RNN	$95.8 \pm 0.1\%$	*87.2 ± 0.9%*	$0.0 + 0\%$	$80.0 \pm 0.2\%$
J-DRAGGN	$96.6 \pm 0.2\%$	*87.9 ± 1.9%*	$20.2 \pm 20.4\%$	$83.7 \pm 2.8\%$
I-DRAGGN	**97.0 ± 0.2%**	$84.9 \pm 1.8\%$	**97.0 + 0.0%**	**94.7 ± 0.5%**

TABLE 3.3: Action-oriented and goal-oriented accuracy results (mean and standard deviation across 3 random initializations) on both the standard and unseen datasets. **Bold** indicates the best model, whereas *italics* denotes models with tied performance.

3.6.2 Results

Language grounding accuracies for our two DRAGGN models, as well as the baseline Single-RNN, are presented in Table 3.3. All three models received the same set of training data, consisting of 2660 low-level action-oriented segments and 693 high-level goal-based sentences. All together, there are 17 unique combinations action-oriented callable units and respective binding arguments, and 6 unique combinations of goal-oriented callable units and binding arguments present in the data. Then, we evaluated all three models on the same set of held-out data, which consisted of 295 low-level segments and 86 high-level sentences.

In aggregate, the models that use callable units for both action- and goal-based language grounding demonstrate superior performance to the Single-RNN baseline, largely due to their ability to generalize, and output combinations unseen at train time. We break down the performance on each task in the following three sections.

3.6.3 Action Prediction

We evaluate the performance of our models on low-level language that directly specifies an action trajectory. An instruction is correctly grounded if the output trajectory specification corresponds to the ground-truth action sequence. To ensure fairness, we augment the output space of Single-RNN to include all distinct action trajectories found in the training data (an additional 17 classes, as mentioned previously).

All models perform generally well on this task, with Single-RNN correctly identifying the correct action callable unit on 95.8% of test samples, while both DRAGGN models slightly outperform with on 96.6% and 97.0% respectively.

3.6.4 Goal Prediction

In addition to the action-oriented results, we evaluate the ability for each model to ground goal-based commands. An instruction is correctly grounded if the output of the grounding module corresponds to the ground-truth (grounded) reward function.

In our domain, all models predict the correct grounded reward function with an accuracy of 84.9% or higher, with the Single-RNN and J-DRAGGN models being too close to call.

3.6.5 Unseen Action Prediction

The Single-RNN baseline model is completely unable to produce unit-argument pairs that were never seen during training, whereas both DRAGGN models demonstrate some capacity for generalization. The I-DRAGGN model in particular demonstrates a strong understanding of each token within the original natural language utterances which, in large part, comes from the separate embedding spaces maintained for callable units and binding constraints respectively.

3.7 Discussion

Our experiments show that the DRAGGN models have a clear advantage over the existing state-of-the-art in grounding action-oriented language. Furthermore, due to the factored nature of the output, I-DRAGGN generalizes well to unseen combinations of callable units and binding arguments.

Nevertheless, I-DRAGGN did not perform as well as Single-RNN and J-DRAGGN on goal-oriented language. This is possibly due to the small number of goal types in the dataset and the strong overlap in goal-oriented language. Whereas the Single-RNN and J-DRAGGN architectures may experience some positive transfer of information (due to the shared parameters in each of the two models), the I-DRAGGN model does not because of its assumed independence between callable units and binding arguments. This ability to allow for positive information transfer suggests that J-DRAGGN would perform best in environments where there is a strong overlap in the instructional language, with a relatively smaller but complex set of possible action sequences and goal conditions.

On action-oriented language, J-DRAGGN has grounding accuracy of around 20.2% while I-DRAGGN achieves a near-perfect 97.0%. Since J-DRAGGN only encodes the input language instruction once, the resulting vector representation

is forced to characterize both callable unit and binding argument features. While this can result in positive information transfer and improve grounding accuracy in some cases (*e.g.* goal-based language), this enforced correlation heavily biases the model towards predicting combinations it has seen before. By learning separate representations for callable units and binding arguments, I-DRAGGN is able to generalize significantly better. This suggests that I-DRAGGN would perform best in situations where the instructional language consists of many disjoint words and phrases.

While our results demonstrate that the DRAGGN framework is effective, more experimentation is needed to fully explore the possibilities and weaknesses of such models. One of the shortcomings in the DRAGGN models is the need for segmented data. We found that all evaluated models were unable to handle long, compositional instructions, such as "Go up three steps, then down two steps, then left five steps". Handling conjunctions of low-level commands requires extending our model to learn how to perform segmentation, or producing sequences of callable units and arguments.

3.8 Conclusion

In this paper, we presented the Deep Recurrent Action/Goal Grounding Network (DRAGGN), a hybrid approach that grounds natural language commands to either action sequences or goal conditions, depending on the language. We presented two separate neural network architectures that can accomplish this task, both of which factor the output space according to the compositional structure of our semantic representation.

We show that overall the DRAGGN models significantly outperform the existing state of the art. Most notably, we show that the DRAGGN models are capable of generalizing to action sequences unseen during training time.

Despite these successes, there are still open challenges with grounding language to novel, unseen environment configurations. Furthermore, we hope to extend our models to handle instructions that are a mixture of goal-oriented and action-oriented language, as well as to long, sequential commands. An instruction such as "go to the blue room, but avoid going through the red hallway" does not map to either an action sequence or a traditional, Markovian reward function. We believe new tools and approaches will need to be developed to handle such instructions, in order to handle the diversity and complexity of human natural language.

Chapter 4

Iterative Language Grounding

This chapter consists of work that was done jointly with Eugene Charniak and Stefanie Tellex[1], currently in submission.

The central theme of this chapter is **interpretability**. Specifically, we show how to reformulate the problem of question-answering on short stories as an iterative language grounding problem, where each sentence of a short story corresponds to updates on an underlying world model. By stating the problem in this way, we show how we can leverage the grounding procedure to perform the structured reasoning necessary for question-answering in an interpretable, reliable way, with minimal amounts of training data and supervision.

4.1 Abstract

We examine the problem of interpreting sequences of natural language utterances, for the goal of understanding, reasoning, and answering questions about text. The challenge is that often, language assumes knowledge about an underlying environment. Existing approaches are limited in that they seek to learn the dynamics of this environment implicitly, from incomplete information. In this work, we address these limitations by presenting the Iterative Grounding Network (IGN), an approach that explicitly factors in the underlying environment by mapping language utterances to concrete updates on an external world model, via reinforcement learning. Our approach allows us to leverage powerful computational abilities for reasoning and deduction inherent in the world model. We show that by factoring in the World Model as an explicit component, the IGN framework achieves near state-of-the-art results on three existing human-robot interaction datasets, with a fraction of the supervision utilized by baselines. Additionally, we show that with fewer than 20 fully annotated examples, we can augment the IGN, allowing it to understand and answer questions about short stories of up to 20 sentences, obtaining near-perfect accuracy on a suite of 20 language grounding tasks based on the 1000 example version of the bAbI Question-Answering Tasks.

[1]Siddharth Karamcheti, Eugene Charniak, Stefanie Tellex: "Iterative Grounding Networks: Grounding Natural Language to World Updates with Minimal Supervision". *In Submission*

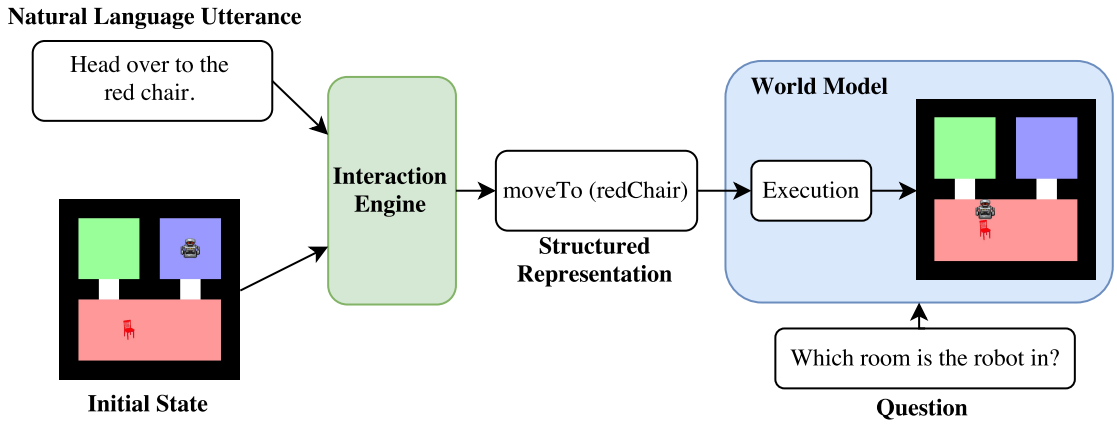


FIGURE 4.1: Iterative Grounding Network Example. Interaction Engine processes utterances, and outputs a structured representation to be executed (e.g. by a robot) to update world state. We can query the world state to answer questions.

4.2 Introduction

Many tasks across human-robot interaction and natural language processing assume a unifying world model, responsible for maintaining the characteristics of an environment. These models encode important features, ranging from static information about the world's attributes, to dynamic information about the relationships between the entities. In human-robot interaction, for example, a possible world model may include the locations of all the objects a robot may interact with, along with the full set of actions it can take. In question-answering on short stories, a possible world model could include other time-dependent information, like the motivations and goals for each character.

In both of these cases, language acts to update the state of the world. Mapping language to these concrete updates then becomes an important problem, as doing so not only grants the ability to execute these updates (i.e. in a human-robot interaction setting), but also reason about the dynamics of a given world (i.e. in a question-answering setting). Furthermore, if these updates are structured in such a way that they can all be performed on some external world model, like a database, state machine, or the physical world, we can fully interpret how a set of utterances changes an environment. In doing so, arbitrary users can gain transparency into the full dynamics, and utilize complex tools and algorithms to perform structured reasoning, and make logical deductions.

Existing approaches for human-robot interaction and question-answering either try to map utterances to a structured representation, treating the rest of the world as a black-box, or try learn the world dynamics in-model, directly from the data. In the case of the former, most approaches require language that is

fully annotated with the ground truth structured representation, which is extremely expensive to collect, and not robust to representational changes. In the case of the latter, most "end-to-end" methods are not easily interpretable, with the current state of the world often existing as some vector representation inside a neural network. Additionally, any structured reasoning needs to be learned, and external tools cannot be effectively leveraged.

In this work, we address these problem by introducing a new framework, the Iterative Grounding Network (IGN), for grounding natural language to structured updates on a pre-existing environment. Iterative Grounding Networks are characterized by a World Model, responsible for explicitly tracking the world state, and an Interaction Engine, that maps language to structured world updates. Because all updates happen external to the learned Interaction Engine, we present a novel reinforcement learning algorithm for training IGN models, given minimal feedback from the world. Namely, we show that we can train IGN models solely given the initial state, the set of (possibly numerous) language utterances to ground, and a weak validation constraint to be run on the final world state. We assess the efficacy of our IGN models through application to three recently proposed datasets for human-robot instruction grounding, as well as a hybrid task combining language grounding and text-based question answering. For the latter, we introduce a new dataset based on the well-known bAbI tasks to fully demonstrate the effectiveness of our framework. The bAbI Tasks test a wide variety of skills, from tracking relations between entities, to reasoning and deduction tasks like counting, positional reasoning, and path finding (Weston et al., 2015), making them the perfect testbed for evaluating the IGN.

On tasks in language grounding for human-robot interaction, we show that the IGN can match state-of-the-art neural network methods, given less information. We also show that because the World Model is factored out of the learning process in the IGN, we can shift the reasoning required for question-answering outside of the model, focusing the learning on the task of mapping language to structured world updates. As a result, we show that given fewer than 20 fully-annotated examples (out of the full training set of 1000 examples), our IGN models can obtain higher than 95% accuracy on 18 of the 20 tasks, matching the performance of upper bound models that were given significantly more information, and significantly beating the performance of end-to-end memory-augmented neural network approaches.

4.3 Related Work

There is a large body of work that examines different methods for grounding language to structured and unstructured representations.

Karamcheti et al. (2017) learn a fully supervised model, the deep recurrent action/goal grounding network (DRAGGN) for mapping human-robot instructions specifying either actions or goal conditions to a factored representation. We utilize the same factored representation in our work, as it is a succinct functional form that grants the ability to handle a wide variety of language in a very efficient manner, unlike other representations like action trajectories or lambda calculus expressions. Other work is concerned with interpreting utterances in weakly supervised settings. Instead of being provided with full annotations, the only supervision signal is based on the final state of the world, after grounding. Artzi and Zettlemoyer (2013) learn a weakly-supervised CCG Parser for navigational instruction following. Williams et al. (2017) also learn a weakly-supervised CCG Parser for instruction grounding, but with a smarter validation function, eliminating the need for explicit planning. More recently, Misra, Langford, and Artzi (2017) use policy gradient methods in a contextual bandit setting with reward shaping to map visual observations and text to actions. Guu et al. (2017) combine REINFORCE (Williams, 1992) with Maximum Marginal Likelihood to map utterances to programs that encapsulate the intent of the language specification. Unfortunately, these newer reinforcement learning methods require significant amounts of data, and tens of thousands of training iterations to converge.

Other work models the world implicitly, via the use of end-to-end neural networks. Most of the following evaluates on the bAbI Tasks (Weston et al., 2015), a set of 20 question-answering tasks that test skills in entity tracking, reasoning, and deduction. End-to-End Memory Networks (Sukhbaatar et al., 2015) are based on the more involved Memory Networks (Weston, Chopra, and Bordes, 2014), and read stories in multiple passes, attending to different sentences with each pass. Henaff et al. (2016) develop the Recurrent Entity Network, a type of Recurrent Neural Network (RNN) consisting of multiple independent modules, each responsible for tracking separate entities of a story, allowing for efficient question answering after only a single pass. Santoro et al. (2017) propose Relation Networks, for explicitly modeling relationships between entities over time. However, such methods fail to be openly interpretable and transparent, because the representation of the world state lives internal to the neural network. Additionally, these models all require a significant number of training examples, and fail to work on smaller datasets (Henaff et al., 2016). Our approach, in contrast, performs all language grounding in the external World Model, in a very transparent, interpretable manner. We also show how we can use our model in low-data situations, with extremely limited supervision.

4.4 Problem Setting

We consider the problem of mapping natural language utterances to explicit updates on a world model, such that after processing, we end up in a desired target state. More concretely, given an example $\mathbf{x} = (s_0, \mathbf{u})$, where s_0 denotes the initial state of the world, and $\mathbf{u} = u_1, u_2, \ldots u_n$ are the series of natural language utterances, we seek to find the best set of world updates $\hat{\mathbf{w}} = \hat{w_1} \ldots \hat{w_n}$ that when applied to the initial state s_0 result in goal state $\mathbf{y} = s_T$. Note that in this work, all world updates w_i are completely deterministic, so $s_T = \mathbf{w}(s_0)$. Furthermore, we assume that each utterance u_i uniquely and independently specifies an update w_i. As such, our formal objective is as follows:

$$\hat{w_1} \ldots \hat{w_n} = \arg\max_{w_1 \ldots w_n} \left[\prod_{i=1}^{n} \Pr(w_i \mid s_{i-1}, u_i) \right] \tag{4.1}$$

We note here that this objective is similar to the way (Guu et al., 2017) formalize their parsing objective. Specifically, we note a similarity between the selection over programs $\mathbf{z}$ based on natural language utterances $\mathbf{u}$ utilized in their semantic parser, and our selection over world updates $\mathbf{w}$.

4.5 Approach

In order to decompose the problem of mapping arbitrary natural language utterances to concrete world updates, we define Iterative Grounding Networks, with the following two components: A World Model, used to track the world state, and an Interaction Engine, responsible for mapping utterances to the corresponding world update.

4.5.1 Iterative Grounding Networks

In this section, we develop the Iterative Grounding Network (IGN) capable of working with externally specified World Models. Because the World Model is external (i.e. it takes the form of some Database, or lives in some code running on a robot), the representation output by the Interaction Engine needs to be structured. Only from this structured representation can the external World Model actually update the world state, to serve as the IGN input at the next time step. Because this hand-off is non-differentiable, and the inputs to the next time step are determined by the external update procedure, it is impossible to train the Interaction Engine in a supervised fashion, with standard back-propagation.

Instead, we reformulate the objective in Eq. 4.1 as a reinforcement learning problem, where at every time step, the Interaction Engine receives a new observation, or utterance u_i, and picks an action w_i to apply to the World Model to maximize the total expected reward (of ending up in the desired goal state). This allows us to use policy gradient methods (Williams, 1992), providing a gradient that can be used to train the Interaction Engine. Specifically, we provide an algorithm based on on Advantage Actor-Critic (A2C) (Mnih et al., 2016) for training our Interaction Engine.

4.5.2 World Model

We assume a World Model $\mathcal{W}$, consisting of a set of entities and objects. Because it is externally specified, we also assume access to the set of structured representations $\mathcal{R}$ used to represent all possible world updates.

While there are many valid structured representations we could use, including lambda calculus, a structured query language like SQL, or even a lightweight functional programming language (Guu et al., 2017), we opt to use the representation utilized by Karamcheti et al. (2017) in their system for language grounding, where all world updates can be parameterized by a single function that takes a variable number of arguments. Not only is this a lightweight representation, but because of it's factored nature (arguments can be shared across different function calls), we can perform very efficient learning. Concretely, we assume access to the set of functions $\mathcal{F}$ and arguments $\mathcal{A}$ such that $\mathcal{R} = \mathcal{F} \times \mathcal{A}$. Arguments include the primitive objects and entities of the underlying world, as well as simple predicates or attributes of the same. Note that in this work, we assume that every world update w consists of a single function call $f \in \mathcal{F}$, called with (possibly several) arguments $\mathbf{a} = a_1 \dots a_n \in \mathcal{A}$. We leave the handling of nested function calls to future work. Given such a parameterization, we can consider the World Model $\mathcal{W}$ as a state machine consisting of a state s, where every function call f with arguments $\mathbf{a}$ transforms the state in a pure, deterministic manner ($s_{t+1} = f(\mathbf{a}, s_t)$).

4.5.3 Interaction Engine Training Objective

We now develop the Interaction Engine objective and training algorithm given this setup of the problem. If we return to the overall objective outlined in Eq. 4.1, we see that:

$$\prod_{i=1}^{n} \Pr(w_i \mid s_{i-1}, u_i) = \prod_{i=1}^{n} \Pr(f_i, \mathbf{a_i} \mid s_{i-1}, u_i) \tag{4.2}$$

Algorithm 1 Parallel A2C with Factored Actions

// Parameters: $\theta_f, \theta_a, \theta_v$ (Policies and Value Function)
// Restore θ_f, θ_a from supervised policy (if pre-training)
Set episode counter $E \leftarrow 1$
Initialize parallel environments **P** with Training Examples
repeat
  Initialize time step $t \leftarrow 1$
  Reset gradients: $d\theta_f \leftarrow 0, d\theta_a \leftarrow 0, d\theta_v \leftarrow 0$
  Create 2D Arrays $\mathbf{s}, \mathbf{u}, \mathbf{v}, \mathbf{r}$ indexed by t, $p \in \mathbf{P}$
  repeat
    Get states, utterances $\mathbf{s_t}, \mathbf{u_t} \leftarrow \mathbf{P}$
    Compute $\pi_f, \pi_a, \mathbf{v_t}$ for all **P** in single pass
    for $p \in \mathbf{P}$ **do**
      if *p is not terminal* **then**
        Sample $f \sim \pi_f[p]$, $\mathbf{a} \sim \pi_a[p]$
        Perform world update $w = (f, \mathbf{a})$
        Receive immediate reward $\mathbf{r_t}[p] \leftarrow r$
      end if
    end for
    $t \leftarrow t + 1$
  until All environments in P are done
  for $p \in \mathbf{P}$ **do**
    $R \leftarrow 0$, $\tilde{A} \leftarrow 0$
    for $i \in t-1, \ldots, 1$ **do**
      // Compute discounted rewards, advantages
      $R \leftarrow \mathbf{r_i}[p] + \gamma R$
      $\tilde{A} \leftarrow \left(\mathbf{r_i}[p] + \gamma \mathbf{v_{i+1}}[p] - \mathbf{v_i}[p]\right) + (\gamma\lambda)\tilde{A}$
      $d\theta_f \leftarrow d\theta_f + \nabla_{\theta_f} \tilde{A} \cdot \log \pi_f(f \mid s_i, u_i; \theta_f)$
      $d\theta_a \leftarrow d\theta_a + \nabla_{\theta_a} \tilde{A} \cdot \log \pi_a(\mathbf{a} \mid s_i, u_i; \theta_a)$
      $d\theta_v \leftarrow d\theta_v + \partial (R - \mathbf{v_i}[\mathbf{p}])^2$
    end for
  end for
  Perform updates of $\theta_f, \theta_a, \theta_v$ using accumulated gradients.
until $E > E_{max}$

If, at training time, we were given fully annotated data of the form $(u_i, f_i, \mathbf{a_i})$, this would be a straightforward objective, learnable via traditional supervised learning techniques. However, we don't have any knowledge about the mapping between utterances and function-argument tuples. Instead, we are only given examples of the form $(s_0, u_1 \ldots u_n, s_T)$, which just tells us the desired goal state of the world.

We can reformulate Eq. 4.2 to work in this setting by treating this as a model-free reinforcement learning problem. Concretely, at each time step t, with the current world state s_t, we receive a new utterance u_t, that informs our selection

over actions, or world state updates $w_t = (f_t, \mathbf{a_t})$. To be explicit, an action in our setup consists of the function f to be called, and the arguments $\mathbf{a}$ it is to be called with. Every time we perform an update, we receive a reward r_t that is dependent on our current state, and the chosen update. In our setup, because we do not have a measure of the quality of performing an update w_t at any intermediate step s_t, the majority of our rewards r_t are 0. Our only non-zero reward comes at the last time step $T-1$, where $r_{T-1} = +1$ if $s_T = w_{T-1}(s_{T-1})$, and is 0 otherwise. To smooth our rewards over all time steps, we decay our reward over the entire time horizon by a discount factor γ, that assigns more reward to recent update actions taken. We refer to the discounted reward received at time step t as R_t.

As such, instead of maximizing the probability of selecting the correct update given each natural language utterance, as in Eq. 4.1, the problem now becomes one of maximizing the total expected reward, across all natural language utterances:

$$\max J(\theta) = \mathbb{E}_{w \sim \pi(w|s,u;\theta)} \big[\sum_t R_t \big] \tag{4.3}$$

where the expectation is over $\pi(w \mid s, u) = \pi(f, \mathbf{a} \mid s, u)$, the probability of generating the update represented by function-argument $(f, \mathbf{a})$ given the utterance u, and the current state of the world s. We represent this policy π with a deep neural network parameterized by weights θ.

One way to optimize this is to use REINFORCE (Williams, 1992):

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\pi(w|s,u;\theta)} \big[\sum_t R_t \big] \\ &= \mathbb{E}_\pi \big[\nabla_\theta \log \pi(w \mid s, u; \theta) R_t \big] \end{aligned}$$

Here, gradient updates are given by the gradient of the logarithm of our policy, scaled by the discounted reward at time t. Unfortunately, a known problem with REINFORCE is high variance, resulting in several training episodes for the policy network to converge. To remedy this, we instead use Advantage Actor-Critic (A2C), where we reformulate the above objective by substituting the discounted reward R_t with an estimate of the advantage A_t (Mnih et al., 2016; Sutton et al., 1999):

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_\pi\big[\nabla_\theta \log \pi(w \mid s, u; \theta) A_t\big] \\
\text{where:} \qquad A_t &= Q_\pi(s, u, w) - V_\pi(s, u) \\
Q_\pi(s, u, w) &= \mathbb{E}_\pi[R_t \mid s, u, w] \\
V_\pi(s, u) &= \mathbb{E}_\pi[R_t \mid s, u]
\end{aligned}$$

Instead of maximizing expected reward, we maximize advantage, the difference of our state-action value function $Q_\pi(s, u, w)$, and a zero-expectation baseline, our state value function $V_\pi(s, u)$. Intuitively, this learns to prioritize update actions that result in better than average reward, and de-prioritize actions resulting in worse than average reward.

Because the actual state-action value function Q_π is unknown, we can estimate it by using our discounted rewards R_t, giving us an estimate of advantage $\tilde{A}_t = R_t - V_\pi(s, u)$, where V_π is learned along with the policy π. However, Schulman et al. (2015) introduce a better estimation for Advantage called Generalized Advantage Estimation (GAE), which we utilize in this work. GAE allows us to control the weight of the state-value function estimate and the weight of recent actions with hyper-parameters, giving us more control over the learning process.

The final piece is the decomposition of our policy $\pi(w \mid s, u; \theta)$ into two independent policies over the functions f and arguments $\mathbf{a}$ (recall that $w = (f, \mathbf{a})$). Sharma et al. (2017) show that any compositional policy $\pi(f, \mathbf{a} \mid s, u)$ in an actor-critic model (like A2C/A3C) can be factored as the product of separate policies $\pi(f \mid s, u)$ and $\pi(\mathbf{a} \mid s, u)$. More importantly, they demonstrate during train time, actions can be chosen by sampling independently from the separate sub-policies, often leading to faster training, and more generalizable behavior . With this in mind, our final Interaction Engine training objective is as follows:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_\pi\big[\nabla_\theta \log \pi(f, \mathbf{a} \mid s, u; \theta)\big] \cdot \tilde{A}_t \\
&= \mathbb{E}_\pi\big[\nabla_\theta \log \big(\pi_f(f \mid s, u; \theta) \pi_a(\mathbf{a} \mid s, u; \theta)\big)\big] \cdot \tilde{A}_t \\
&= \mathbb{E}_\pi\big[(\nabla_{\theta_f} \log \pi_f) + (\nabla_{\theta_a} \log \pi_a)\big] \cdot \tilde{A}_t
\end{aligned}$$

We present a variant of the A2C Algorithm for training this objective with the factored action space in Algorithm 1.

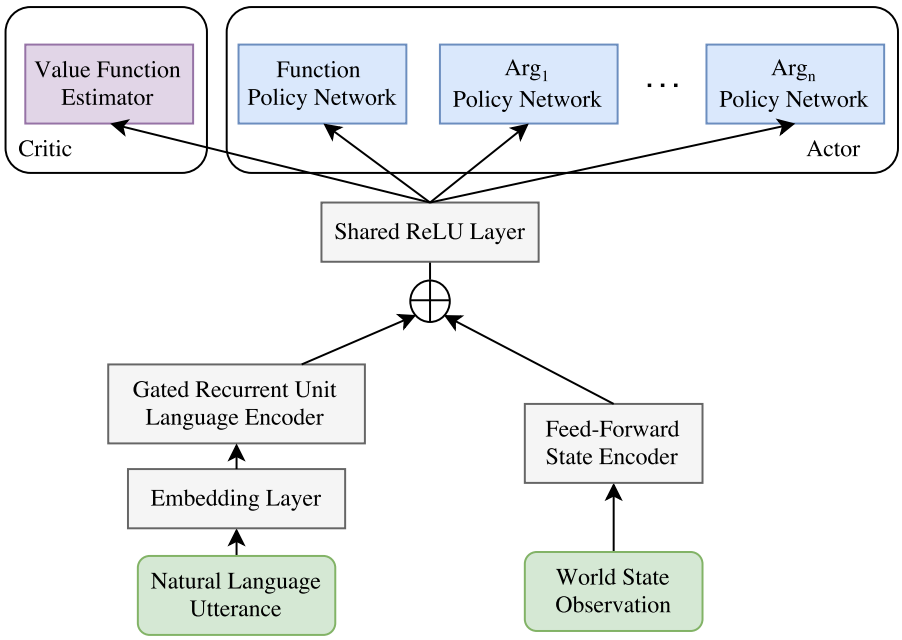


FIGURE 4.2: Iterative Grounding Network Architecture.

4.5.4 Interaction Engine Neural Architecture

We implement our Interaction Engine with the neural network architecture shown in Figure A.1. The Interaction Engine takes as input at every time step t the natural language utterance u_t, as well as a representation of the current world state s_t. The natural language utterance u_t is first embedded via an embedding layer, where the embeddings start off as random. The variable length utterance embeddings are then mapped into a fixed size vector with a Gated Recurrent Unit (Cho et al., 2014), a type of Recurrent Neural Network cell popular in many natural language processing tasks, like machine translation, question answering, as well as prior work in human-robot interaction (Cho et al., 2014; Karamcheti et al., 2017).

The state representation s_t can be represented arbitrarily, and encoded with a corresponding architecture. For example, if the state were represented as an image, a Convolutional Neural Network (CNN) architecture would be appropriate. However, here we assume that the state s_t is represented as a vector. We encode it via a two-layer feed-forward network, with the ReLU activation function. The encoded state and utterance are then concatenated, and fed to a shared hidden layer, with another ReLU. The resulting vector is then fed to multiple sub-networks, for generating the state-value function estimate V_t, as well as the function policy $\pi_f(f \mid s, u)$, and the argument policy $\pi_a(\mathbf{a} \mid s, u)$.

In the case of functions with multiple arguments, we allow for multiple argument sub-networks. Each sub-network consists of a single hidden layer with

a ReLU activation, followed by the output layer. In the case of the state-value estimate, this final layer has a linear activation, while each of the policy sub-networks have a final softmax activation, to generate probability distributions over functions/arguments. To regularize the network, we use Dropout (Srivastava et al., 2014). We train our network using the Adam Optimizer (Kingma and Ba, 2014), with a learning rate of .00001. All models are implemented in Tensorflow (Abadi et al., 2015).

4.5.5 Training with Annotated Examples

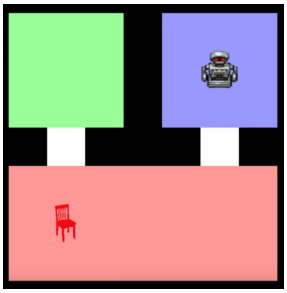
In many scenarios, we may have access to a small number of fully annotated examples, where each utterance is annotated with the exact function/argument structured update tuple. Especially in scenarios where there are a large number of possible function/argument tuples, or where there are several utterances in a row without a reward signal, including this small amount of information can significantly learning.

To formalize this, we assume access to a small number of annotated examples (u, s, w), where $w = (f, \mathbf{a})$. With this, we now have a well-defined supervised learning objective – namely, maximize the probability of generating $f, \mathbf{a}$ from the given state s and utterance u. Equivalently, we minimize the sum of the cross-entropy losses between the predicted distributions $\pi(f \mid s, u)$, $\pi(\mathbf{a} \mid s, u)$ and the true labels $f, \mathbf{a}$.

At train time, we perform this training steps by initially pre-training the policy networks $\pi(f), \pi(\mathbf{a})$ to convergence on the given set of annotated examples. We then train the rest of the IGN via Reinforcement Learning, using the procedure outlined in Algorithm 1. To further improve the stability of the training procedure, every several thousand iterations of the reinforcement learning procedure, we run a very small number of supervised training steps on the annotated examples. We find empirically that this helps prevent catastrophic forgetting. We train the fully-supervised pipeline with Dropout (Srivastava et al., 2014), and the Adam Optimizer (Kingma and Ba, 2014), with a learning rate of .001.

4.6 Experiments

We evaluate the IGN on two applications: understanding natural language instructions in a human-robot interaction setting, and question-answering on short stories. The first application shows that with even in cases of a single instruction, the IGN enables training with less supervision than prior approaches. The second application shows that the IGN is able to leverage the World Model effectively, synthesizing information from multiple sentences.



(A) Cleanup World

	Examples	Funcs	Args	Tasks
WeakSup	500	1	5	5/5
Goals	779	3	5	6/15
Actions	2955	4	7	15/28

(B) Cleanup World Dataset Statistics.

FIGURE 4.3: Cleanup World Language Grounding Dataset Statistics

4.6.1 Cleanup World Language Grounding

The first set of experiments are conducted on the Cleanup World Mobile Manipulator Domain (MacGlashan et al., 2015), pictured in Figure 4.3a. The Cleanup World domain consists of a robot agent, one or more objects, and three rooms of different colors. Possible robot actions involve fine-grained actions, like moving some steps in a certain direction , to more general actions like having the agent move an object between rooms. We utilize datasets put forth by recent language grounding approaches. Statistics describing the datasets can be found in Figure 4.3b.

The first Cleanup World Experiment (*WeakSup*) utilizes the dataset introduced by Williams et al. (2017) in their work on weakly-supervised CCG Semantic Parsing. The second and third Cleanup World experiments (*Goals*, *Actions*) utilize the datasets presented by Karamcheti et al. (2017), for grounding action-oriented and goal-oriented language. These datasets consist of single natural language utterances annotated with the states of the world before and after grounding. Even though we only ground a single utterance before validation, this is still a critical task, as the instruction language is rather ambiguous and complex. A model that works in such a setting would benefit the human-robot interaction community, as this weakly supervised data is easier to collect, while also being transferable to other representations. To validate the structured representation output by our IGN, we use the procedure developed by Williams et al. (2017) that returns $+1$ reward if the given structured representation results in the desired final state, and 0 otherwise. Full details about how we utilized these datasets in our experiments can be found in the supplemental material.

To fairly evaluate the IGN, we limit ourselves to baselines that utilize the same function/argument representation (as the complexity of the structured representation significantly affects grounding performance). As such, in addition to IGN results, we present the DRAGGN results on each of the datasets, as an upper bound (UB). The DRAGGN model (Karamcheti et al., 2017) gets strictly more data than the IGN as it is trained in a fully supervised setting. In this way, it

	Random	IGN	DRAGGN (UB)
WeakSup	20%	93.0%	93.0%
Goals	16.67%	84.8%	86.0%
Actions	6.67%	*71.9%*	95.9%

TABLE 4.1: Cleanup World Results. Results use model with highest validation accuracy across three seeds.

provides a meaningful upper bound, as it is given the ground truth groundings that the IGN is trying to learn.

Results

Table 4.1 has the IGN and DRAGGN upper bound results for the three different Cleanup World experiments. On the Weakly Supervised dataset (*WeakSup*), the IGN is able to perfectly match the upper bound performance, obtaining a near perfect 93.0% grounding accuracy, given the small number of examples. We note that this is especially interesting, as the validation function the IGN is provided with contains strictly less information than the fully annotated utterances the DRAGGN model is provided with. On the Goal-Oriented language corpus (*Goals*), the IGN is able to achieve close to the DRAGGN performance, obtaining a final grounding accuracy of 84.8% (compared to the maximum possible 86.0%). This dataset contains a task output space slightly larger than the previous Weakly Supervised dataset, and contains significantly more complex language. Again, we note the IGN's ability to match DRAGGN performance given significantly less information. However, on the Action-Oriented corpus (*Actions*), we see that the IGN has performance that fails to match DRAGGN, by a large margin (71.9% accuracy vs. 95.9% accuracy). Upon thorough error analysis, we found this was due to mode collapse, as the original dataset is very imbalanced (of the 2955 examples, 2264 all have the same argument label).

We note this as an interesting failure mode, and hope to perform more experiments in future work.

4.6.2 Hybrid bAbI Grounding and Question-Answering

The second set of experiments are conducted on a new dataset, based on the 20 well-known synthetic bAbI Question-Answering Tasks (Weston et al., 2015). In the original bAbI dataset, each task consists of 1000 story, question, and answer tuples, where stories consist of several sentences (for our work, all stories are truncated to be no longer than 20 sentences). Note that we utilize the 1000 (1k) version of the dataset, rather than the 10,000 example (10k) version utilized

by most recent work (Henaff et al., 2016; Santoro et al., 2017). We do this to show that IGN models can be effective given small amounts of data, unlike the cited approaches. The tasks test varying skills, from tracking relations between multiple entities, to the ability to deduce or induce missing information, to the ability to reason about size, position, or geographical location. Because of this wide variety of well-defined tasks, the bAbI dataset makes a good testbed for the evaluation of language grounding models.

The full procedure used to convert the original bAbI dataset into the Hybrid dataset can be found in the supplemental material. We find that we can represent a given task's world updates with functions from up to 4 classes, and up to three arguments from up to 14 classes. Stories are limited to be 20 sentences long, which means that in many cases, the IGN has to choose world updates from the set of possible functions and arguments over a horizon of 10-20 separate examples without a reward signal. From this, it is impractical to assume that a random policy will learn to properly ground such stories without a significant amount of train time. However, we find that if we start with a small number of annotated examples, we can significantly accelerate learning. As such, we test two versions of the IGN, one where it is provided with 10 fully annotated stories (only 1% of the total dataset), and another where it is provided with 20 stories (only 2%). The full Hybrid bAbI QA/Language Grounding Dataset annotated with the full set of function/argument tuples can be found at the following URL: https://sites.google.com/site/winsupplemental/.

We implement the World Model as a state machine (implemented in code) that consumes world updates and updates its internal fields in a pure and consistent manner. Some example updates include agentToRoom(entity, location) that moves a specified person to the given location, link(entity, object), for picking up an object, and isA(entity, property) for assigning characteristics to an entity. Question-Answering is done on the final state, after all updates have been processed. Any reasoning that is necessary for answering the question is also handled by the World Model. For example, if we have a question like "How many objects is Mary carrying," we have a function (in code) that explicitly counts the objects that have been linked to Mary by the set of world updates, rather than having to track the quantity implicitly, as is done in existing methods. As mentioned previously, this ability to use external tools to perform reasoning is a strong benefit of our approach.

To put the IGN results in context, we present a series of lower and upper bounds. Again, this is because any strict evaluation is based on the complexity of the underlying update representation. We present two lower bounds (LB), each of which receive strictly less information than the IGN. These neural models only receive story, question, answer tuples, and are trained end-to-end. The first lower bound we present is a simple LSTM approach, while the second lower bound model is an End-to-End Memory Network (MemN2N) (Sukhbaatar et

	LSTM (LB)	MemN2N (LB)	IGN + 10	IGN + 20	DRAGGN (UB)	MemNN (UB)
1 Supporting Fact	50.0	100.0	**100.0**	100.0	100.0	100.0
2 Supporting Facts	20.0	91.7	**96.9**	99.8	99.8	100.0
3 Supporting Facts	20.0	59.7	**96.7**	96.7	96.7	100.0
2 Arg Relations	61.0	97.2	*95.8*	**100.0**	100.0	100.0
3 Arg Relations	70.0	86.9	*53.7*	*85.7*	*85.8*	98.0
Yes/No Questions	48.0	92.4	**98.3**	100.0	100.0	100.0
Counting	49.0	82.7	83.1	**95.6**	99.8	85.0
Lists/Sets	45.0	90.0	*57.1*	**96.5**	100.0	100.0
Simple Negation	64.0	86.8	92.1	**100.0**	100.0	100.0
Indefinite Knowledge	44.0	84.9	90.7	**95.1**	99.2	98.0
Basic Coreference	62.0	99.1	**99.5**	100.0	100.0	100.0
Conjunction	74.0	99.8	**100.0**	100.0	100.0	100.0
Compound Coreference	94.0	99.6	*93.0*	*95.8*	100.0	100.0
Time Reasoning	27.0	98.3	*92.3*	**100.0**	100.0	99.0
Basic Deduction	21.0	100.0	**100.0**	100.0	100.0	100.0
Basic Induction	23.0	98.4	**100.0**	100.0	100.0	100.0
Positional Reasoning	51.0	49.0	51.9	69.0	90.1	65.0
Size Reasoning	52.0	88.9	**96.6**	99.6	99.6	95.0
Path Finding	8.0	17.2	79.7	**100.0**	100.0	36.0
Agent Motivation	91.0	100.0	**100.0**	100.0	100.0	100.0
Solved Tasks ($>$ 95%)	0	8	11	**18**	18	17

TABLE 4.2: Hybrid bAbI Results. IGN Models are trained via RL, with 10 (1%), and 20 (2%) of the stories fully annotated. *Italics* denote cases where IGN results are worse than MemN2N, while **Bold** denotes best IGN model that solves the task ($>$ 95% accuracy).

al., 2015) that explicitly reads the story via multiple passes, then finally generates the answer. The two upper bounds receive strictly more information than the IGN. The first upper bound, the DRAGGN, receives fully annotated stories. We note that this is the most meaningful model for interpreting the IGN results, as it directly bounds the results in the same manner as in the prior experiments. The second upper bound is the Memory Network with Supporting Fact Supervision (MemNN) (Weston, Chopra, and Bordes, 2014). The Memory Network gets the original sentences of the story, along with the set of sentences necessary to answering a question. Because the subset of sentences necessary to answer are significantly smaller than the total number of sentences in the story, the MemNN model has the best performance on the original dataset at the cost of needing the most annotated data.

Results

Table 4.2 has the results for both versions of the IGN, as well as all lower and upper bounds. Notably, we show that providing just 10 annotated examples is enough to gain greater than 95% accuracy on 11 of the 20 tasks. If we increase the number to 20 annotated stories, we solve 18 of the hybrid tasks. In other words, solely by providing the World Model, and a very small number of annotated examples (2% of the total training set), we can obtain near-perfect performance.

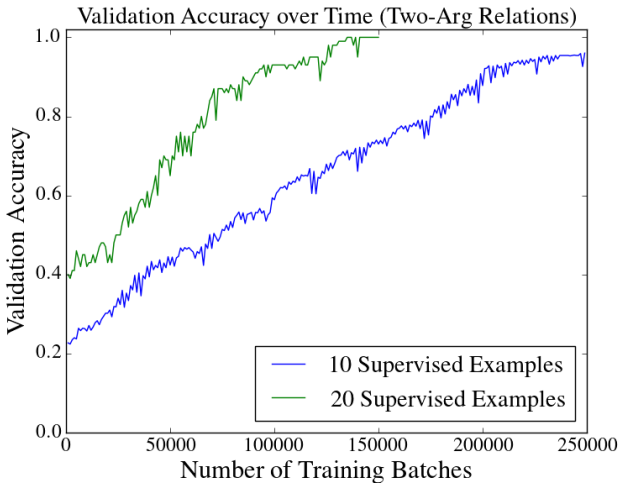


FIGURE 4.4: Learning curves plotting accuracy on validation set vs. number of training iterations for the Two-Arg Relations Hybrid bAbI Task.

Figure 4.4 shows learning curves on the "2 Arg Relations" Task, one of the only tasks requiring more than 20,000 training steps. Note that with more annotated examples, the IGN learns faster and more efficiently.

4.7 Discussion

Our evaluation shows that not only is the IGN able to successfully operate in the context of human-robot interaction, as per our Cleanup World experiments, but also in the context of story understanding, solving 18 out of the 20 Hybrid bAbI question tasks.

On Cleanup World, we find that most of the time, the IGN is able to match (or almost match) the upper bound DRAGGN state-of-the-art results. This seems to strongly indicate the effectiveness of the IGN, as it takes minor performance hits, with the major benefit of requiring less data to train. Not only is it significantly easier to collect data of the form required for the IGN (namely utterance, initial state, and final state), but it allows for the use of many different types of structured representations with the same dataset. In the case of the fully supervised DRAGGN models, the language utterances all must be annotated with the required function/argument pairs, which is not only extremely expensive, but not robust to representational changes.

While the results of the Cleanup World experiments are strong, the crucial benefits of the IGN are found in the Hybrid bAbI results. Most notable are the cases

where the IGN is able to significantly outperform the End-to-End Memory Network lower bound, for example, on the tasks of "Counting", "Indefinite Knowledge", and "Path Finding". The key similarity between each of these three tasks is the need for relatively complex reasoning and deduction to come up with the answer given the story. For example, in the "Path Finding" task, stories consist of descriptions of the world (i.e. "The kitchen is north of the hallway, the garden is east of the hallway"), and questions asking to find paths between certain locations (i.e. "How do I get from the kitchen to the garden?"). This requires complex reasoning to go from the story information describing the world, to the required path (i.e. "south, east"). In the case of the End-to-End Memory Network, because this reasoning is happening internal to the neural network model, it is hard to generate the correct answer, as the network is not structured to perform the explicit logical steps. However, in the case of the IGN, because the World Model is specified externally (namely, in code), any set of tools (in our case, a simple search algorithm) can be leveraged to find the given path from the story information.

Also important are the cases where the IGN fails to outperform the End-to-End Memory Network lower bound, namely, on the "3 Arg Relations" and "Compound Coreference" Tasks. We note that the key similarity between these tasks is that they all require world updates that are represented with a single function, and exactly three arguments (the maximum number of arguments we needed across all tasks). For the "3 Arg Relations" task, this amounts to a total number of world updates equal to $4 \cdot 5 \cdot 14 \cdot 14 = 3,920$ possible options. While the IGN is clearly able to learn how to map utterances to the correct world state, it is not able to do so efficiently, as in some sense, it is trying to solve a harder problem than is necessary (both these tasks have a large amount of unnecessary information in the stories). Because the End-to-End Memory Network is given the question a priori, it can effectively search over the entire story and attend to only the important parts, while the IGN has to ground every utterance with high accuracy, which in these cases prove to be a harder task.

In other words, because the IGN factors out the World Model, we can not only effectively ground language for human-robot interaction, but also answer questions for text-based story understanding. The external model allows us to not only fully interpret and understand the grounding procedure, but also lets us use a wide variety of specialized tools and algorithms for performing reasoning and deduction, something that would be impossible with an end-to-end model. Best of all, we can obtain strong results with the IGN, either matching or exceeding the existing state-of-the-art, with significantly less data and supervision.

While the external World Model can clearly lead to significant benefits, it is limiting because it cannot be directly learned. Not only does the World Model need to be expertly designed, but the process of adding additional literals or actions may require non-trivial tweaks. As such, scaling to new domains would necessitate

an updated World Model, capable of handling new update actions. However, we argue that the interpretability, transparency, and ability to perform structured reasoning provided by the World Models exceed the cost, especially in the use-cases we have outlined above. We hope to address some new applications of the IGN, as well as some of the aforementioned problems in future work.

4.8 Conclusion

We present the Iterative Grounding Network (IGN), a framework for mapping language to structured world updates. We develop the IGN using a deep neural network to map language utterances and state observations into a representation based on functions and arguments. We also present a novel reinforcement learning algorithm based on Advantage Actor-Critic (A2C) for training the IGN, where the only inputs are the initial state of the world, the set of utterances to ground, and a final validation constraint to run on the final state. Finally, our results show that we can use the IGN to great effect in applications to tasks in human-robot interaction, as well as text-based question answering. On the latter, the IGN grants full transparency into the grounding process, allowing for the use of external tools to perform structured reasoning, solving 18 of the 20 Hybrid bAbI Tasks.

While the IGN is a step in the right direction, there remain open challenges. One challenge would be to learn to compose primitive functions to represent nested behaviors, without the need for additional hand-engineered forms. These "macro-functions" would allow for the generalization to more tasks, with less information. Another possible direction would be to explore the capabilities of the external world model. Because you can use arbitrary tools in tandem with the world model, it would be interesting to see the possibilities of the IGN framework in combination with an external database, to do more general question answering.

Chapter 5

Conclusion: Looking Forward

The goal of this thesis is to explore the problem of language grounding by examining three themes in language – **abstraction**, **generalization**, and **interpretability** – and their application to tasks in human-robot interaction and question-answering. In the first chapter, we present work on hierarchical language grounding, where we develop a system capable of handling abstraction in language, grounding both high-level instructions and low-level instructions in a single model, for the purposes of faster and more accurate grounding to robot behavior. In the second chapter, we present work on generalizing across actions and goals, where we develop a new suite of models and representations capable of mapping language to either goal representations or actions, as well as generalize to unseen tasks. Finally, in the third chapter, we examine interpretability, mapping the language in short stories to concrete database updates, allowing for complex language reasoning to happen in an understandable way.

While the key punchlines of this work are the above themes, there are two others that are equally important, and referenced throughout this work – **modularity** and **representation**. To build truly effective language grounding systems, both of these are necessary components: a language grounding system needs to be modular, capable of interfacing with a series of tools and systems to generate behavior (like the motion planners and databases in this work). Furthermore, a language grounding system needs to trade in the right form of representation, a structure that allows one to express a large variety of different tasks, with minimal hand-engineering or domain-specificity.

While the results in this thesis are promising in regards to the aforementioned themes, there is still a long way to go. The work in this thesis is limited for one reason: the language interaction is one-sided. Users give instructions, a model reads a story, and then outputs a behavior. To truly bridge the gap between human users and complex systems, language grounding needs to take **interaction** into account. Grounding models needs to be able to communicate back to users, asking questions, and gleaning information to allow for better generalization.

We ground language because we envision a world without a gap between everyday people and complex technology – where everything is truly accessible. This thesis is a first step towards that goal, but there is so much more to do.

Appendix A

Iterative Grounding Details

A.1 Interaction Engine Neural Architecture

We implement our Interaction Engine with the neural network architecture shown in Figure A.1. The Interaction Engine takes as input at every time step t the natural language utterance u_t, as well as a representation of the current world state s_t. The natural language utterance u_t is first embedded via an embedding layer, where the embeddings start off as random. The variable length utterance embeddings are then mapped into a fixed size vector with a Gated Recurrent Unit (Cho et al., 2014), a type of Recurrent Neural Network cell popular in many natural language processing tasks, like machine translation, question answering, as well as prior work in human-robot interaction (Cho et al., 2014; Karamcheti et al., 2017).

The state representation s_t can be represented arbitrarily, and encoded with a corresponding architecture. For example, if the state were represented as an image, a Convolutional Neural Network (CNN) architecture would be appropriate. However, here we assume that the state s_t is represented as a vector. We encode it via a two-layer feed-forward network, with the ReLU activation function. The encoded state and utterance are then concatenated, and fed to a shared hidden layer, with another ReLU. The resulting vector is then fed to multiple sub-networks, for generating the state-value function estimate V_t, as well as the function policy $\pi_f(f \mid s, u)$, and the argument policy $\pi_a(\mathbf{a} \mid s, u)$.

In the case of functions with multiple arguments, we allow for multiple argument sub-networks. Each sub-network consists of a single hidden layer with a ReLU activation, followed by the output layer. In the case of the state-value estimate, this final layer has a linear activation, while each of the policy sub-networks have a final softmax activation, to generate probability distributions over functions/arguments. To regularize the network, we use Dropout (Srivastava et al., 2014). We train our network using the Adam Optimizer (Kingma and Ba, 2014), with a learning rate of .00001. All models are implemented in Tensorflow (Abadi et al., 2015).

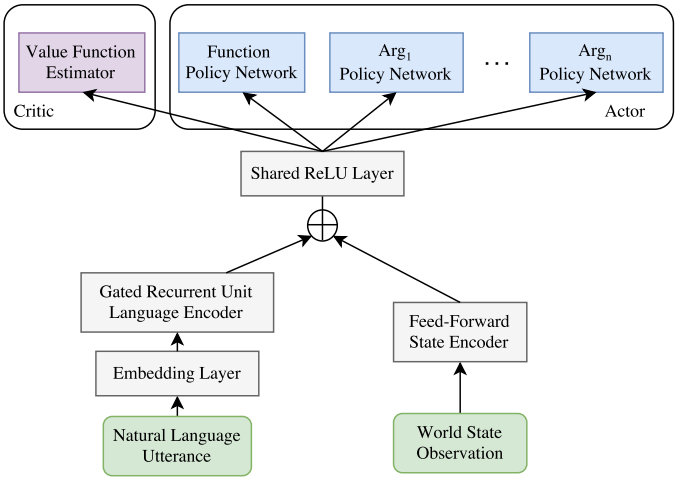


FIGURE A.1: Iterative Grounding Neural Network Architecture.

A.2 Experiments

A.2.1 Cleanup World Language Grounding

The first Cleanup World Experiment (*WeakSup*) utilizes the dataset introduced by Williams et al. (2017) in their work on weakly-supervised CCG Semantic Parsing. The dataset presented is small, containing only 500 examples, spanning 5 different tasks. The tasks all involve the robot agent moving to a specific location, either a room or an object. Each example consists of a single utterance, a set of different pre-condition states of the world prior to grounding, as well as a set of post-condition states that are all satisfiable given the correct grounding and respective pre-condition state. Williams et al. (2017) additionally define a validation function that outputs 1 or 0 based on if the produced CCG parse satisfies all post-condition states given each pre-condition state of the example in question.

To apply the IGN to this dataset, we simplify the lambda calculus parse language utilized in the original work into a language of functions and arguments, identical to the representation used by Karamcheti et al. (2017). This structured representation consists of a single function, and five different arguments. We additionally modify the presented validation function to take in the modified functional language. As such, the IGN setup involves feeding in the utterance and initial states, generating the function/argument tuple, and then calculating the reward (via the validation function), based on if the respective update satisfies each post-condition state of the example.

The second and third Cleanup World experiments (*Goals*, *Actions*) utilize the datasets presented by Karamcheti et al. (2017), in their work on grounding action-oriented and goal-oriented language. The authors present two datasets, one

consisting of goal-oriented language, and the other consisting of action-oriented language. Because Karamcheti et al. (2017) work in a fully supervised context, each example consists of the utterance, the initial state of the world, and the correct function/argument tuple to generate. To turn this into a weakly supervised problem, such that we can illustrate the effectiveness of the IGN, we create a validation function similar to that of Williams et al. (2017), by executing each function/argument tuple to get the final post-condition state of the world. We then discard the ground truth function/arguments that are supposed to be generated, and train our IGN solely using the raw utterances and validation function.

We train the IGN only via Reinforcement Learning, for 1600 iterations of Algorithm 1. The initial states of the Cleanup World domain are represented as hot-encoded vectors that store the location of the robot agent. We use an embedding size of 30, an RNN encoder size of 128, processing 8 separate utterances per forward pass.

A.2.2 Hybrid bAbI Grounding and Question-Answering

To convert the original bAbI Question-Answering dataset into the Hybrid bAbI Question-Answering/Language Grounding dataset, we first define a set of functions and arguments that encapsulate the full set of world updates that are represented for each of the 20 tasks. We find that on average, each task's set of world updates can be represented with around a function from up to 4 classes, and up to three arguments from up to 14 classes. We then modify the original code used to generate the tasks (Weston et al., 2015), to annotate each sentence of the original bAbI QA tasks (version 1.2) with the corresponding world state update (function/argument tuple). Note that the stories in our dataset are exactly the same as those in the original (we are not generating new stories). To generate the final validation constraint, we turn each of the question/answer pairs from the original dataset into a boolean function, that returns 1 reward if True given the final state of the world, and 0 reward otherwise. More precisely, if we are given a question like "Where is Mary?" with the answer "kitchen", our validation function returns 1 reward if and only if in the final world state, Mary ends up in the kitchen. As mentioned in the paper, the full Hybrid bAbI QA/Language Grounding Dataset annotated with the full set of function/argument tuples can be found at the following URL: `https://sites.google.com/site/winsupplemental/`.

We implement the World Model as a state machine (implemented in code) that consumes a given world update and updates its internal fields in a pure and consistent manner. For example, given an utterance like "Mary went to the kitchen" which maps to the function/arguments agentToRoom(Mary, kitchen), the state machine merely updates its internal dictionary storing locations to have the key "Mary" point to the location "kitchen." This is a simple procedure that is not

only efficient at modeling the full state of the world, but is effective to query at validation time. Any reasoning that needs to happen for validation is also executed in code, which is a crucial benefit of our approach. For example, if we have a question like "How many objects is Mary carrying," we have a function (in code) that explicitly counts the objects that have been linked to Mary by the given set of world updates, rather than having to track the quantity implicitly, as is done in existing end-to-end methods.

We specifically use such reasoning to great effect in the following reasoning-focused tasks: "Counting", "Lists/Sets", "Indefinite Knowledge", "Basic Deduction", "Basic Induction", and "Path Finding." In the "Path Finding" task specifically, in which we report significantly better results than the lower bound model, we found this reasoning to be extremely important. More precisely, we used a simple dynamic programming algorithm to take stories (of the form "The garden is east of the hallway," "The kitchen is north of the hallway") and questions (of the form "How do I get from the kitchen to the garden") and compute the actual path necessary for the answer (namely, "south, east"). Having this algorithm responsible for the structured reasoning allowed for perfect performance on this task, whereas all end-to-end methods failed.

Tables 1 and 2 (next page) give exhaustive dataset statistics, and provide the full list of all world update functions utilized for the Hybrid bAbI Tasks.

We train the IGN via the hybrid supervised learning/reinforcement learning procedure outlined in the Approach section, for up to 250,000 iterations of Algorithm 1, with early stopping after 20,000 iterations if the accuracy on the validation set exceeds 95% (we find only a small number of tasks need more than 20,000 iterations). We perform an initial pre-training of 120 supervised iterations on the limited number of annotated examples. Initial states are represented by hot encoding any information relevant to the task (i.e. entity locations in the entity relation tasks, presence of objects in reasoning tasks) in a vector. To further improve the stability of the training procedure, every several thousand iterations of the reinforcement learning procedure, we run a very small number of supervised training steps on the annotated examples. We find empirically that this helps prevent catastrophic forgetting. To this end, we perform 20 supervised iterations on the limited number of annotated examples every 2,000 reinforcement learning iterations. We use an embedding size of 30, with an RNN size of 128, processing 8 separate utterances per forward pass.

	Max Story Length	Functions	Argument 1	Argument 2	Argument 3
1 Supporting Fact	10	1	5	7	-
2 Supporting Facts	20	3	5	10	-
3 Supporting Facts	20	3	5	10	-
2 Arg Relations	2	4	7	7	-
3 Arg Relations	20	4	5	14	14
Yes/No Questions	20	3	5	10	-
Counting	20	4	5	14	-
Lists/Sets	20	3	5	10	-
Simple Negation	10	1	5	7	-
Indefinite Knowledge	10	2	5	7	7
Basic Coreference	10	1	7	7	-
Conjunction	10	1	7	5	5
Compound Coreference	10	2	7	6	5
Time Reasoning	14	4	5	7	-
Basic Deduction	8	2	9	5	-
Basic Induction	9	2	6	9	-
Positional Reasoning	2	4	7	7	-
Size Reasoning	15	2	7	7	-
Path Finding	5	4	7	7	-
Agent Motivation	12	3	5	12	-

TABLE A.1: Dataset Statistics for the Hybrid bAbI Tasks Dataset

Function Signature	Description
agentToRoom(entity, location)	Move an entity to the given location.
link(entity, object)	Link an object to an entity (for picking up objects).
unlink(entity, object)	Unlink an object and entity (for dropping objects).
isNorthOf(room1, room2)	Specify that room1 is north of room2 (positional reasoning).
isEastOf(room1, room2)	Specify that room1 is east of room2 (positional reasoning).
isWestOf(room1, room2)	Specify that room1 is west of room2 (positional reasoning).
isSouthOf(room1, room2)	Specify that room1 is south of room2 (positional reasoning).
transfer(person1, person2, object)	Transfer object from person1 to person2.
addMotive(person, motive)	Add motivation (hungry, thirsty, etc.) to given person.
fitsInside(object1, object2)	Specify that object1 fits inside object2 (size reasoning).
isBiggerThan(object1, object2)	Specify that object1 is bigger than object2 (size reasoning).
isA(entity, characteristic)	Assign given characteristic to entity.
isColor(entity, color)	Assign given color to entity.
isAfraidOf(entity1, entity2)	Specify that entity1 fears entity2.
agentToRoomYesterday(entity, location)	Specify that entity went to location yesterday.
agentToRoomMorning(entity, location)	Specify that entity went to location this morning.
agentToRoomAfternoon(entity, location)	Specify that entity went to location this afternoon.
agentToRoomEvening(entity, location)	Specify that entity went to location this evening.
agentsToRoom(entity1, entity2, location)	Move entity1 and entity2 to location.
agentsMaybeIn(entity1, entity2, location)	Specify that entities might be in location.

TABLE A.2: Exhaustive Set of Functions defined for Hybrid bAbI Tasks.

Bibliography

Abadi, Martín et al. (2015). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *CoRR* abs/1603.04467.

Andreas, Jacob and Dan Klein (2015). "Alignment-based compositional semantics for instruction following". In: *Conference on Empirical Methods in Natural Language Processing*.

Artzi, Yoav and Luke Zettlemoyer (2013). "Weakly supervized learning of semantic parsers for mapping instructions to actions". In: *Annual Meeting of the Association for Computational Linguistics*.

Arumugam, Dilip et al. (2017). "Accurately and Efficiently Interpreting Human-Robot Instructions of Varying Granularities". In: *CoRR* abs/1704.06616.

Bellman, R. (1957a). "A Markovian decision process". In: *Indiana University Mathematics Journal* 6 (4), pp. 679–684.

Bellman, Richard (1957b). *Dynamic Programming*. 1st ed. Princeton, NJ, USA: Princeton University Press.

Bengio, Yoshua et al. (2000). "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research* 3, pp. 1137–1155.

Brooks, Daniel J. et al. (2012). "Make it so: Continuous, Flexible Natural Language Interaction with an Autonomous Robot". In: *AAAI Conference on Artificial Intelligence Workshop on Grounding Language for Physical Systems*.

Brown, Peter F. et al. (1990). "A Statistical Approach to Machine Translation". In: *Computational Linguistics* 16, pp. 79–85.

Brown, Peter F. et al. (1993). "The Mathematics of Statistical Machine Translation: Parameter Estimation". In: *Computational Linguistics* 19, pp. 263–311.

Chen, David L. and Raymond J. Mooney (2011). "Learning to Interpret Natural Language Navigation Instructions from Observations." In: *AAAI Conference on Artificial Intelligence*.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Empirical Methods in Natural Language Processing*.

Chung, Junyoung et al. (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555.

Dieterrich, Thomas G. (2000). "Hierarchical reinforcement learning with the MAXQ value function decomposition". In: *Journal on Artificial Intelligence Research* 13, pp. 227–303.

Diuk, Carlos, Andre Cohen, and Michael L. Littman (2008). "An object-oriented representation for efficient reinforcement learning". In: *International Conference on Machine Learning*.

Dzifcak, Juraj et al. (2009a). "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution". In: *IEEE International Conference on Robotics and Automation*.

Dzifcak, Juraj et al. (2009b). "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution". In: *IEEE International Conference on Robotics and Automation*.

Google Speech API (2017). https://cloud.google.com/speech/. Accessed: 2017-01-30.

Gopalan, Nakul et al. (2016). "Planning with Abstract Markov Decision Processes". In: *International Conference on Machine Learning Workshop on Abstraction in Reinforcement Learning*.

– (2017). "Planning with Abstract Markov Decision Processes". In: *International Conference on Automated Planning and Scheduling*.

Guu, Kelvin et al. (2017). "From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood". In: *CoRR* abs/1704.07926.

Hemachandra, Sachithra et al. (2015). "Learning Models for Following Natural Language Directions in Unknown Environments". In: *IEEE International Conference on Robotics and Automation*.

Henaff, Mikael et al. (2016). "Tracking the World State with Recurrent Entity Networks". In: *CoRR* abs/1612.03969.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9, pp. 1735–1780.

Howard, Thomas M., Stefanie Tellex, and Nicholas Roy (2014). "A natural language planner interface for mobile manipulators". In: *IEEE International Conference on Robotics and Automation*.

Iyyer, Mohit et al. (2015). "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Conference of the Association for Computational Linguistics*.

Jong, Nicholas K. and Peter Stone (2008). "Hierarchical model-based reinforcement learning: R-max + MAXQ". In: *International Conference on Machine Learning*.

Junghanns, Andreas and Jonathan Schaeeer (1997). "Sokoban: a Challenging Single-agent Search Problem". In: *International Joint Conference on Artificial Intelligence Workshop on Using Games as an Experimental Testbed for AI Reasearch*.

Karamcheti, Siddharth et al. (2017). "A Tale of Two DRAGGNs: A Hybrid Approach for Interpreting Action-Oriented and Goal-Oriented Instructions". In: *CoRR* abs/1707.08668.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980.

MacGlashan, James et al. (2015). "Grounding English commands to reward functions". In: *Robotics: Science and Systems*.

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

Matuszek, Cynthia et al. (2012). "Learning to parse natural language commands to a robot control system". In: *International Symposium on Experimental Robotics*.

McGovern, Amy, Richard S. Sutton, and Andrew H Fagg (1997). "Roles of macro-actions in accelerating reinforcement learning". In: *Grace Hopper Celebration of Women in Computing* 1317.

McMahan, H. Brendan, Maxim Likhachev, and Geoffrey J. Gordon (2005). "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees". In: *International Conference on Machine Learning*.

Mei, Hongyuan, Mohit Bansal, and Matthew R. Walter (2016). "Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences". In: *AAAI Conference on Artificial Intelligence*.

Mikolov, Tomas et al. (2010). "Recurrent neural network based language model". In: *Interspeech*.

Mikolov, Tomas et al. (2011). "Extensions of recurrent neural network language model". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing*.

Mikolov, Tomas et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781.

Misra, Dipendra, John Langford, and Yoav Artzi (2017). "Mapping Instructions and Visual Observations to Actions with Reinforcement Learning". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Vancouver, Canada: Association for Computational Linguistics.

Mnih, Volodymyr et al. (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *ICML*.

Paul, Rohan et al. (2016). "Efficient Grounding of Abstract Spatial Concepts for Natural Language Interaction with Robot Manipulators". In: *Robotics: Science and Systems*.

Puterman, Martin L. (1994). "Markov Decision Processes: Discrete Stochastic Dynamic Programming". In:

Quigley, Morgan et al. (2009). "ROS: an open-source Robot Operating System". In: *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*.

Raman, Vasumathi and Hadas Kress-Gazit (2011). "Analyzing Unsynthesizable Specifications for High-Level Robot Behavior Using LTLMoP". In: *International Conference on Computer-Aided Verification*.

Reed, Scott E. and Nando de Freitas (2016). "Neural Programmer-Interpreters". In: *International Conference on Learning Representations*.

Santoro, Adam et al. (2017). "A simple neural network module for relational reasoning". In: *CoRR* abs/1706.01427.

Schulman, John et al. (2015). "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *CoRR* abs/1506.02438.

Sharma, Sahil et al. (2017). "Learning to Factor Policies and Action-Value Functions: Factored Action Space Representations for Deep Reinforcement learning". In: *CoRR* abs/1705.07269.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Sukhbaatar, Sainbayar et al. (2015). "End-To-End Memory Networks". In: *NIPS*.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215.

Sutton, Richard S., Doina Precup, and Satinder P. Singh (1999). "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112, pp. 181–211.

Sutton, Richard S. et al. (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *NIPS*.

Tellex, Stefanie et al. (2011). "Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation". In: *AAAI Conference on Artificial Intelligence*.

Vogel, Adam and Dan Jurafsky (2010). "Learning to follow navigational directions". In: *Annual Meeting of the Association for Computational Linguistics*.

Weston, Jason, Sumit Chopra, and Antoine Bordes (2014). "Memory Networks". In: *CoRR* abs/1410.3916.

Weston, Jason et al. (2015). "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks". In: *CoRR* abs/1502.05698.

Williams, Edward C. et al. (2017). "Learning to Parse Natural Language to Grounded Reward Functions with Weak Supervision". In: *AAAI Fall Symposium on Natural Communication for Human-Robot Collaboration*.

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8, pp. 229–256.

Yamada, Tatsuro et al. (2016). "Dynamical Linking of Positive and Negative Sentences to Goal-Oriented Robot Behavior by Hierarchical RNN". In: *International Conference on Artificial Neural Networks*.