Grounding Natural Language Instructions to Semantic Goal Representations for Abstraction and Generalization

Dilip Arumugam* Siddharth Karamcheti* Nakul Gopalan Edward C. Williams
Mina Rhee Lawson L.S. Wong Stefanie Tellex

December 8, 2017

Abstract

Language grounding is broadly defined as the problem of mapping natural language instructions to robot behavior. To truly be effective, these language grounding systems must be accurate in their selection of behavior, efficient in the robot's realization of that selected behavior, and capable of generalizing beyond commands and environment configurations only seen at training time. One choice that is crucial to the success of a language grounding model is the choice of representation used to capture the objective specified by the input command. Prior work has been varied in its use of explicit goal representations, with some approaches lacking a representation altogether, resulting in models that infer whole sequences of robot actions, while other approaches map to carefully constructed logical form representations. While many of the models in either category are reasonably accurate, they fail to offer either efficient execution or any generalization without requiring a large amount of manual specification. In this work, we take a first step towards language grounding models that excel across accuracy, efficiency, and generalization through the construction of simple, semantic goal representations within Markov decision processes. We propose two related semantic goal representations that take advantage of the hierarchical structure of tasks and the compositional nature of language respectively, and present multiple grounding models for each. We validate these ideas empirically with results collected in a simulated mobile-manipulator domain, as well as demonstrations of a physical robot responding to spoken instructions in real time. Our grounding models tie abstraction in language commands to a hierarchical planner for the robot's execution, enabling a response-time speed-up of several orders of magnitude over baseline planners within sufficiently large domains. Concurrently, our grounding models for generalization infer elements of the semantic representation that are subsequently combined to form a complete goal description, enabling the interpretation of commands involving novel combinations never seen during training. Taken together, our results show that the design of semantic goal representation has powerful implications for the accuracy, efficiency, and generalization capabilities of language grounding models.

1 Introduction

As robots become ubiquitous, there is a greater need for a simple yet effective means of interaction between them and their human collaborators. Natural language is a powerful tool that facilitates seamless interaction between humans and robots. Not only does it offer flexibility and familiarity, but it also eliminates the need for end users to have knowledge of low-level programming. The challenge, however, lies in dealing with the full scope of natural language in all of its complexity and diversity. The end goal is for any human to instruct their robot collaborator through language as if they were interacting with another human. To this end, we address the problem of accurate and efficient language grounding along two specific dimensions of language complexity: abstraction and generalization.

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 subgoals 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."

*First authors contributed equally

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. Moreover, this notion of abstraction in language parallels abstraction in sequential decision-making Gopalan et al. [2017]. A language grounding model that properly identifies linguistic abstraction can then use hierarchical planning effectively, enabling more efficient robot execution. Thus, in this work, we think of abstraction as the key to achieving efficiency.

End users of a fully-deployed language grounding systems will maintain an implicit expectation of generalization. For instance, consider a robot operating in the home alongside a user issuing a command like "Take the silverware to the kitchen." Assuming that this command has been seen by the grounding model at training time, it should be straightforward to correctly ground and complete the task. Suppose that the robot is also aware of the dining room within its environment and is subsequently given the command "Take the silverware to the dining room." Despite never having seen this or any other command that specifically involves the dining room and silverware elements of the environment together, the robot should be able to generalize and combine its prior understanding of moving the silverware with its existing knowledge of the dining room's dynamics to successfully complete the task. Notice that without such generalization capabilities, a language grounding model would require a training corpus of commands large enough to capture the full combinatorial space of environment variables and the various possible relationships between them. Alternatively, a robotic system that can associate semantic meaning with the individual environment components incident to each language command has a much greater chance of being able to generalize from a significantly smaller amount of training data.

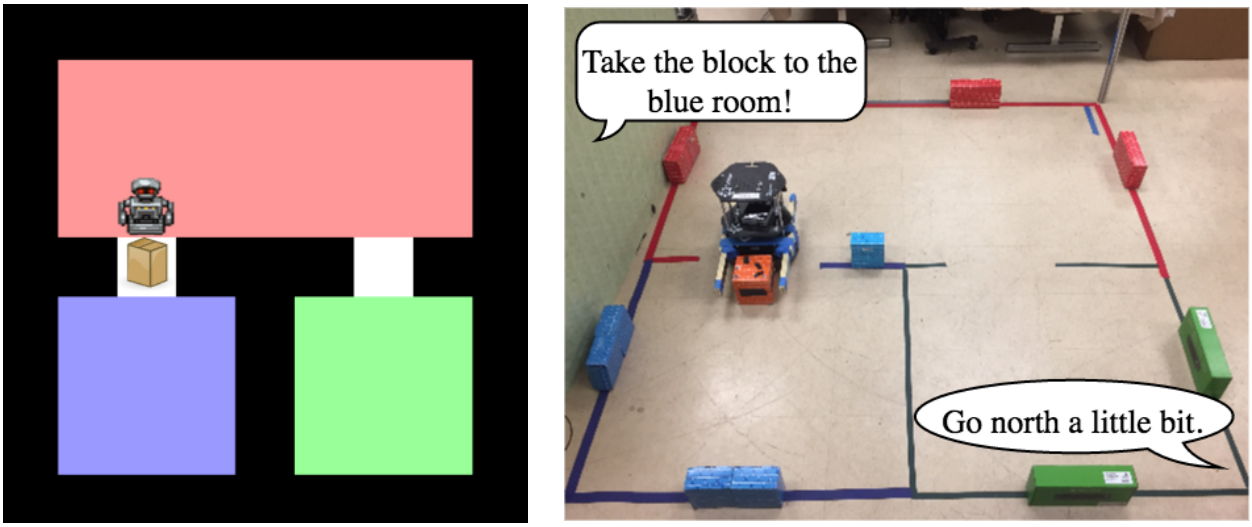
Existing approaches for language grounding map between natural language commands and task representations that are essentially sequences of primitive robot actions [Chen and Mooney, 2011, Matuszek et al., 2012, Tellex et al., 2011]. While effective at directing robots to complete predefined tasks, mapping 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 grounding model can transfer to other robots with different action sets so long as there is consistency in the task representation (in this work, reward functions). The desire to scale up to larger, more complex environments, however, reveals an inherent trade-off within MDPs between accommodating low-level task representations for flexibility and decreasing the time needed to plan through higher level representations of the decision-making problem [Gopalan et al., 2017]. In parallel, performing a single inference to arrive at the intermediate reward function representation compromises the opportunity to associate semantic meaning with individual components of that representation.

To address these problems, we present two novel approaches for mapping natural language commands to robot behavior. The first approach tackles the varying granularity of natural language by mapping 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. By coupling abstraction-level inference with the overall grounding problem, we exploit the subsequent hierarchical planner to efficiently execute the grounded tasks. The second approach tackles the issue of generalization through grounding models that factor the output space and compose reward functions through a multi-step inference process. Rather than inferring the entire semantic representation all at once, these grounding models decompose the representation and infer its constituent elements. As a result, the approach is able to attribute meaning to specific components of the semantic representation, enabling it to exhibit a higher degree of generalization at testing time over baseline approaches.

Portions of this work previously appeared in Arumugam et al. [2017] and Karamcheti et al. [2017]. This work differs in that it presents a unified view of the two systems and, additionally, adds new results highlighting the ability of models presented in Karamcheti et al. [2017] to generalize to novel goal-oriented commands.

2 Related Work

While the various environments in which robots can be found are expanding to include the home, the workplace, and on the road, the most common interfaces for controlling these robots remain fixed in either teleoperation or directly programmed behaviors. Humans use natural language to communicate ideas, motivations, and goals with other humans. It has been a long-standing goal in the field of artificial intelligence to have an agent understand and perform



(a) Simulation of Cleanup world.

(b) Cleanup world with a Turtlebot agent.

Figure 1: Sample configurations of the Cleanup world task on the Turtlebot mobile manipulator and on the simulated environment. (a): Simulated Cleanup world used to collect data for this work. Moreover, the simulator allows us to test out framework before runs on the robot. (b): Examples of high-level (goal-oriented) and fine-grained (action-oriented) commands issued to the Turtlebot robot in a mobile-manipulation task.

tasks that are specified through natural language. SHRDLU [Winograd, 1971] is one of the earliest examples of such an agent wherein handwritten rules enabled the understanding and grounding of language commands to behavior using simple programs. Despite this early success, however, the field cannot rely on methods that require the enumeration of all possible grammer rules.

One alternative methodology maps language commands directly to the action sequences that an agent must execute to realize the task[Tellex et al., 2011, Matuszek et al., 2012, Artzi and Zettlemoyer, 2013]. Tellex et al. [2011] learn the probabilistic meaning of individual words given natural language and associated trajectories. Given a sentence at testing time, inference is performed over the space of possible trajectories to identify one that best grounds the task specified by the input. Matuszek et al. [2012] learn to translate natural language to a synthetic control language that can execute directly within the environment. Artzi and Zettlemoyer [2013] learn a mapping of language to an event-semantics-based representation of the trajectories. The core restrictions of the aforementioned methods are that they limit the space of possible tasks to explicit robot action sequences and heavily rely on the determinism of the underlying environment.

Another line of language grounding research first maps language to an intermediate semantic representation that can be converted to agent behavior [Zelle and Mooney, 1996, Zettlemoyer and Collins, 2005, Howard et al., 2014, MacGlashan et al., 2015]. Examples of semantic representations include lambda calculus expressions, reward functions, and constraints for the agent to satisfy in the environment. Semantic parsing translates natural language into a lambda calculus expression using a grammar that is executable by an agent [Liang, 2016]. Zelle and Mooney [1996] leverage this approach to enable the execution of linguistic queries on a geographical database. Zettlemoyer and Collins [2005] solve the same problem but begin with a partial lexicon and learn new grammar rules during the course of learning the semantic parser. We note that by diverging from the semantic parsing approaches, our approach does not require a manually pre-specified grammar.

Howard et al. [2014] tackle motion planning problems in the robotics domain by learning the meaning of words while mapping from natural language to planner constraints. This method extends ideas from Tellex et al. [2011] where inference is performed over the smaller space of constraints rather than the space of all possible trajectories. Paul et al. [2016] learn the meaning of abstract concepts like rows and columns of objects allowing for commands that capture a large space of possible behaviors. MacGlashan et al. [2015] proposed grounding natural language commands to reward functions associated with certain tasks, leaving the challenge of execution in a potentially stochastic environment to an arbitrary planning algorithm. They treat the goal reward function as a conjunction of propositional logic functions, much like a machine language, to which a natural language task can be translated, using an IBM Model 2 (IBM2)

[Brown et al., 1990, 1993] language model. Additionally, reward functions can themselves be learned via inverse reinforcement learning [Ng and Russell, 2000] using trajectories from expert trainers.

Crucially, MacGlashan et al. [2015] 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 argument. In doing this, the output space of the grounding 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 re-learn the language understanding components for each environment. In order to leverage this flexibility, all of the proposed grounding models in this work produce lifted reward functions which are then completed by a grounding module before being passed to a planner (see Section 5).

Planning in domains with large state-action spaces is computationally expensive as algorithms like value iteration and bounded real-time dynamic programming (RTDP) need to explore the domain at the lowest, "flat" level of abstraction [Bellman, 1957, McMahan et al., 2005]. Naïvely, 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 that an agent can identify and select in sequence. A common method to describe subtasks is through temporal abstraction in the form of macro-actions [McGovern et al., 1997] or options [Sutton et al., 1999]. These methods achieve subgoals using either a fixed sequence of actions or a policy with fixed initial and terminal states respectively. Planning with options requires computing the individual option policies 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 where each subtask is associated with a subgoal and a state abstraction relevant to achieving the subgoal. Unfortunately, both MAXQ and R-MAXQ still suffer from the challenges of bottom-up planning as they must back up the reward from individual action executions across the hierarchy.

We use AMDPs in this paper because they plan in a "top-down" fashion, where a planner first determines how good a subgoal before solving the more computationally intensive problem of planning to achieve the subgoal. Consequently, the planner restricts its computation to a minimal subset of all the subgoals within the hierarchy. AMDPs offer model-based hierarchical representations in the form of a reward function and transition function to each subtask. Specifically, the AMDP hierarchy is an acyclic graph in which each node is a primitive action or an AMDP that solves a subtask defined by its parent; the edges are actions of the parent AMDP. Depending on where a subgoal AMDP resides in the hierarchy, its state space may either consist of low-level environment states (at the lowest level) or abstract state representations built on top of the environment state (at any point above the ground level). AMDPs have empirically been shown to achieve faster planning performance than other hierarchical methods.

The previous language grounding methods presented either map language to a semantic representation or a direct trajectory. However, Dzifcak et al. [2009] posited that natural language can be interpreted as *both* a goal state specification and an action specification. While this idea is orthogonal to the idea of using abstractions, it is equally important as humans often mix between goal-based commands, that specify some desired target state, and action-oriented commands, that provide step-by-step instruction on exactly how to perform a task. To fully cover the space of possible language commands, grounding models ought to be agnostic in realizing the specified behavior. To address this, in our work, we put forth a single representation capable of capturing the semantics of both types of commands.

In this work, we use deep neural networks 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, 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 representations of raw data [Bengio et al., 2000, Mikolov et al., 2013]. In the context of NLP, these representations, or "embeddings", are often high-dimensional vectors that not only represent individual words (similar to sparse word representations) but also capture the semantics of the words themselves. NLP has also enjoyed in the use of recurrent neural networks (RNNs) that contain specially constructed cells for mapping variable length inputs (in this work, language commands) to a fixed-size vector representation [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 models to map between natural language and MDP reward functions.

3 Background

We consider the problem of mapping from natural language to robot behavior within the context of Markov decision processes. A *Markov Decision Process* (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.

We specify an *Object-oriented Markov Decision Process* (OO-MDP) to model the robot's environment and actions [Diuk et al., 2008]. 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 Figure 1b 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. A true value results in a reward signal of $+1$, and 0 otherwise. Using these propositional functions as reward functions that encode termination conditions for each task, we arrive at a sufficient semantic representation for grounding language. In our framework, as in MacGlashan et al. [2015], we wish to map from natural language to propositional reward functions that correctly encapsulate the behavior indicated by the input command; we then have a fully-specified MDP that can be solved with a planning algorithm to produce robot behavior.

In the previous example, the propositional function does not generalize well to different environment configurations; notice that the correspondence between room1 and the blue room is specific to that particular instance of the training environment. If the environment configuration changes at test time, a model trained to map to environment-specific outputs would fail to produce the correct behavior. To remedy this problem, we "lift" the previous propositional functions 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 Section 5.3) that consumes these lifted reward functions and performs the actual low-level binding to specific room instances, resulting in grounded propositional functions like blockInRoom block0 room1.

We evaluate all models on the Sokoban-inspired Cleanup World domain [Junghanns and Schaeeer, 1997, MacGlashan et al., 2015]. The domain consists of a mobile-manipulator agent in a 2-D world with uniquely colored rooms and movable objects. A domain instance is shown in Figure 1a. The domain itself is implemented as an OO-MDP where domain objects include rooms and interactable "blocks" (like the chair, basket, etc.) all of which have location and color attributes.

4 Semantic Goal Representations

We follow the high-level structure of a language grounding system defined by MacGlashan et al. [2015]. In particular, we treat the language grounding problem as a decomposition into task inference and task execution components. During task inference, we are given a natural language command c, and wish to identify the semantic representation $\hat{r}$, such that:

$$\hat{r} = \arg\max_{r} \Pr(r \mid c) \tag{1}$$

Consequently, given a parallel corpus $\mathcal{C}$ at training time, we wish to find a set of model parameters $\hat{\theta}$ such that

$$\hat{\theta} = \arg\max_{\theta} \prod_{(c,r)\in\mathcal{C}} \Pr(r \mid c, \theta) \tag{2}$$

At inference time, given a language command c, we find the best $\hat{r}$ that maximizes the probability $\Pr(r \mid c, \hat{\theta})$. In all models, r is a lifted propositional function that is independent of any environment. Accordingly, this lifted task representation must be passed directly to the grounding module (see Section 5.3) in order to arrive at an environment-representation that can complete the MDP and be used for planning the task execution. In the following sections, we

define two types of semantic goal representations, each of which allows for more efficient execution and generalization respectively.

4.1 Abstraction in Language

When contemplating the question of how to make a robot's execution of a language command more efficient, one possible consideration is that humans use language at multiple levels of abstraction. For example, as illustrated in Figure 1b, a natural language command like "Take the block to the blue room" specifies a high-level task relative to a command like "Go north a little bit" that is trying to exercise more fine-grained control. To support fine-grained tasks, MDPs need high-resolution state and action spaces; however, completing high-level tasks then requires long action sequences, which makes planning inefficient. In contrast, hierarchical planning frameworks, are designed with abstraction in mind and attempt to leverage the inherent hierarchical nature of a task in order to reduce the time needed to solve complicated planning problems [Gopalan et al., 2017]. We propose that one semantic goal representation for efficient execution combines the latent granularity l of a natural language command with a lifted reward function r in a hierarchical planning framework. Through this representation, we empower the system to accurately interpret and quickly execute both abstract and fine-grained commands.

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 this work, we leverage the AMDP hierarchical planning framework that enables efficient execution through the incremental planning of subtasks without solving the full planning problem in its entirety. Finally, we assume that each command is generated from a single level of abstraction (that is, no command supplied to the grounding model can represent a mixture of abstraction levels).

We can now reformulate the learning problem outlined in Equation 1. Given a natural language command c, we find the corresponding level of the abstraction hierarchy l, and the lifted reward function r that maximizes the joint probability of l, r given c. Concretely, we seek the level of the state-action hierarchy $\hat{l}$ and the lifted reward function $\hat{r}$ such that:

$$\hat{l}, \hat{r} = \arg\max_{l,r} \Pr(l, r \mid c) \tag{3}$$

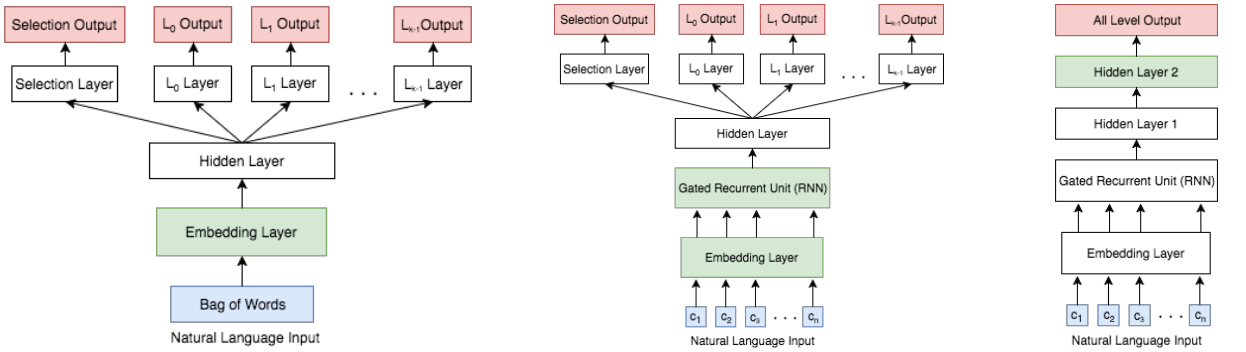
4.2 Factored Reward Functions

While a language command consists of multiple words, each having varied importance on the underlying task specification, the output of our language grounding models thus far has been a singleton reward function. That is, the command "Take the block to blue room," maps to a single output blockInRoom block0 roomIsBlue. We could, however, treat the output in a more compositional manner and attempt to ground substrings in the natural language to individual semantic tokens. For example, blockInRoom is uniquely specified by the substring "block to ... room." Similarly, block0 and roomIsBlue are entirely identified by "the block" and "blue room" respectively. At testing time, when presented with a novel command like "Take the block to the green room," the fact that we've seen the command "Take the block to blue room" during training should be sufficient for us to infer the blockInRoom block0 piece of the output. Unfortunately, a singleton reward function representation compromises our ability to generalize and reason about the constituent elements of an output reward function.

For this reason, we turn to an alternate semantic representation that factors the output space (reward function). We define the *callable unit*, which takes the form of a (possibly multiple) argument function. These functions are paired with *binding arguments* whose possible values depend on the type and arity of the callable unit. In the previous example, blockInRoom would be treated as a callable unit with two separate binding arguments block0 and roomIsBlue. A language grounding model that uses this semantic representation must first infer the callable unit and then infer the constituent binding arguments. By forcing the grounding model to be responsible for making these inferences separately, we allow the model to generalize to novel commands that require the composition of concepts seen in isolation during training.

By reformulating Equation 1, we arrive at a new objective function for this factored output representation. Given a natural language command c, our goal is to find the callable unit $\hat{u}$ and binding arguments $\hat{\mathbf{a}}$ that maximize the following joint probability:

$$\hat{u}, \hat{\mathbf{a}} = \arg\max_{u,\mathbf{a}} \Pr(u, \mathbf{a} \mid c) \tag{4}$$



(a) Multi-NN Model (b) Multi-RNN Model (c) Single-RNN Model

Figure 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, green boxes denote significant structural differences between models.

5 Language Grounding Models

Here we outline two distinct classes of language grounding models, each of which maps between a natural language command and a lifted reward function (propositional function) of an OO-MDP. We conclude the section with a brief description of the Grounding Module which consumes a lifted reward function (that contains no environment-specific entities) and maps to the appropriate grounded reward function that can be passed to an MDP planner.

5.1 Singleton Reward Functions & Abstraction

We compare four language models for identifying the most likely reward function and level of granularity: 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 information regarding the collection procedure and format of the datasets used to train all models, please refer to Section 6.2. For detailed descriptions and implementations of all the presented models, and associated datasets, please refer to the supplemental repository: https://github.com/h2r/GLAMDP.

5.1.1 IBM Model 2

As a baseline, we formulate task inference 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, 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 Equation 3 by Bayes' rule):

$$\hat{l}, \hat{r} = \arg\max_{l,r} \Pr(l, r) \cdot \Pr(c \mid l, r) \tag{5}$$

This task inference formulation follows directly from MacGlashan et al. [2015] and we follow in an identical fashion for training the IBM2 using the standard expectation-maximization (EM) algorithm.

5.1.2 Neural Network Language Models

We develop three classes of neural network architectures (see Figure 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**).

5.1.3 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 Equation 3 as $\Pr(l, r \mid c) = \Pr(l \mid c) \cdot \Pr(r \mid l, c)$. Applying this to the corpus likelihood (Equation 2) and taking logarithms, the Multi-NN objective is to find parameters $\hat{\theta}$:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(\vec{c},l,r)\in\mathcal{C}} \log \Pr(l \mid \vec{c}, \theta) + \log \Pr(r \mid l, \vec{c}, \theta) \tag{6}$$

To learn this set of parameters, we use the architecture shown in Figure 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 a probability distribution with support on the reward functions at level L_i of the hierarchy.

To train the model, we minimize the sum of the cross-entropy loss on each term in Equation 6. 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 $p = 0.5$.

5.1.4 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, 2011, Sutskever et al., 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 $s = \langle c_1, c_2 \ldots c_n \rangle$. The Multi-RNN objective is then:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(s,l,r)\in\mathcal{C}} \log \Pr(l \mid s, \theta) + \log \Pr(r \mid l, s, \theta) \tag{7}$$

This modification is reflected in Figure 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], while requiring fewer parameters than the long short-term memory (LSTM) cell [Hochreiter and Schmidhuber, 1997].

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 Equation 7, 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.

5.1.5 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 r given the natural language command c as $\Pr(l, r \mid c) = \Pr(l \mid c) \cdot \Pr(r \mid l, c)$, 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 (that is, separate parameters for each level of abstraction in the underlying hierarchical planner).

Alternatively, we can directly estimate the joint probability $\Pr(l, r \mid c)$. 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, r), where l denotes the level of abstraction, and r denotes the lifted reward

function at the given level. The Single-RNN objective is to find $\hat{\theta}$ such that:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(s,l,r)\in\mathcal{C}} \log \Pr(l, r \mid s, \theta) \tag{8}$$

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 Figure 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 level-reward tuple (l, r), given the natural language command c.

To train the model, we minimize the cross-entropy loss of the joint probability term in Equation 8. Training hyper-parameters are identical to Multi-RNN, and Dropout is applied to the initial embedding layer and the penultimate hidden layer.

5.2 Factored Reward Functions

While our Single-RNN model manages to generalize over syntax, it makes no attempt to leverage the compositional structure of language and generalize over semantics. A unit-argument pair not observed at training time will not be predicted, even if the constituent pieces were observed separately. To remedy this, the Single-RNN model would require every possible unit-argument pair to be enumerated in the training data to adequately model the full output space. Consequently, the requisite amount of data grows rapidly with task specifications that include more objects with richer attributes.

To resolve this, we introduce the Deep Recurrent Action/Goal Grounding Network (DRAGGN) framework. The DRAGGN framework maps natural language instructions to *separate* distributions over callable units and (possibly multiple) binding constraints, where the callable units generate 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) [Reed and de 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 each architecture in Figure 3, each with its own output space). In this work, 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. At the cost of extending training time, it is straightforward to extend our models to handle extra arguments by adding extra Binding Argument Networks.

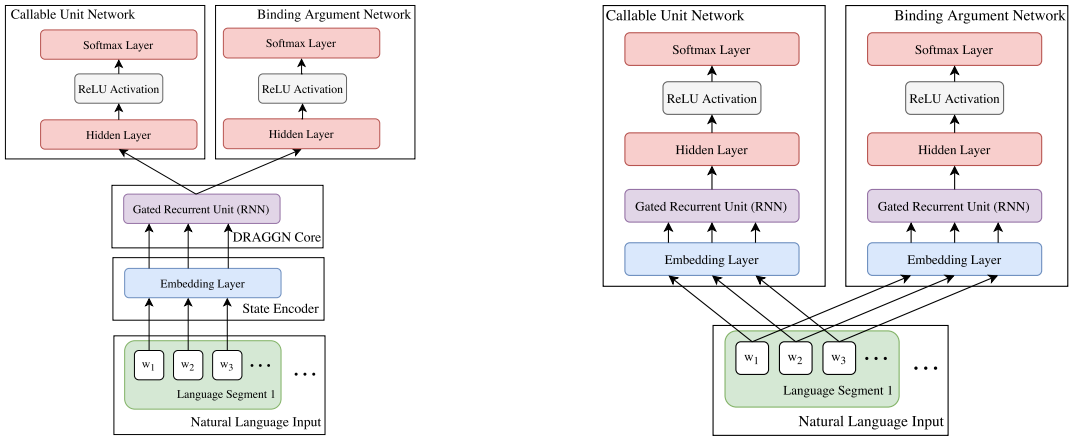
Recall the DRAGGN objective that is given exactly by Equation 4:

$$\hat{u}, \hat{\mathbf{a}} = \arg\max_{u,\mathbf{a}} \Pr(u, \mathbf{a} \mid c) \tag{9}$$

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 data 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 (that is, optimizing with respect to the predicted callable units and binding arguments). At inference, 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. Note that we do not enforce this validity restriction at training time.

Our models were trained using Adam, for 125 epochs, with a batch size of 16, and a learning rate of 0.001.



(a) Joint DRAGGN

(b) Independent DRAGGN

Figure 3: Architecture diagrams for the two Deep Recurrent Action/Goal Grounding Network (DRAGGN) models, introduced in Sections 5.2.1 and 5.2.2. Both architectures ground arbitrary natural language instructions to callable units and binding arguments.

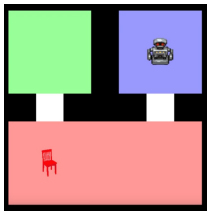
5.2.1 Joint DRAGGN (J-DRAGGN)

The J-DRAGGN models the joint probability in Equation 4, via the shared RNN state in the DRAGGN Core (as depicted in Figure 3a). 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. 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 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.

5.2.2 Independent DRAGGN (I-DRAGGN)

The I-DRAGGN, contrary to the J-DRAGGN, decomposes the objective from Equation 4 by treating callable units and binding arguments as being independent, given the original natural language instruction. More precisely, rather than directly estimating the joint probability of the callable unit c and the arguments $\mathbf{a}$ given the language instruction (as J-DRAGGN does), the I-DRAGGN assumes c and $\mathbf{a}$ are independent given the language instruction. This change is realized in the network architectural differences between the J-DRAGGN and the I-DRAGGN. The I-DRAGGN network architecture is shown in Figure 3b. 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.



(a) A starting instance of the Cleanup World domain.

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

(b) Example AMT commands and corresponding reward functions.

Figure 4: Amazon Mechanical Turk (AMT) dataset domain and examples.

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

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
Head left for a step.	goLeft	2

Table 1: (a) Complete set of action and goal-oriented callable units for the Cleanup World domain. (b) Examples of natural language phrases and corresponding callable units and arguments.

5.3 Grounding Module

In all of our models, the inferred lifted reward function template must be bound 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 Figure 1b), 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 (for instance, 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 categorical grammar (CCG) semantic parser, which grounds ambiguous predicates and nested arguments.

6 Experiments

In this section we outline the series of experiments conducted to evaluate the ability of our grounding models to accurately infer the task specified in natural language while also enabling either efficient execution or generalization. We begin by briefly recalling the Cleanup World domain before describing the abstraction hierarchy defined over the domain as well as the set of callable units and binding arguments defined over the domain. From there we provide details of how our dataset was compiled before presenting the experimental details and results.

6.1 Domain

Recall that the Cleanup World domain [Junghanns and Schaeeer, 1997, MacGlashan et al., 2015], illustrated in Figure 1a, is a mobile-manipulator robot domain that is partitioned into rooms (denoted by unique colors) with open doors (white cells). Each room may contain some number of objects which can be moved (pushed) by the robot. This domain itself 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 in a home environment. We use an AMDP from Gopalan et al. [2017], 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 (specifically, the 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]. Figure 4b shows a few collected sample commands at each level and the corresponding level-specific AMDP reward function.

For factored reward functions, goal-oriented callable units 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 5.3) to produce grounded reward functions (OO-MDP propositional logic functions) that are handled by an MDP planner. 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 are shown in Table 1.

6.2 Dataset

To conduct our evaluation of efficient execution by leveraging abstraction in language, we performed 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 Figure 4a. 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 parallel corpora for treating language grounding as a machine translation problem with ground truth reward functions and abstraction levels. 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 such as "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 (for example, "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.

To conduct our evaluation of generalization capability in the DRAGGN models, we re-use the previously specified dataset of natural language commands for the single instance of Cleanup World domain seen in Figure 1a. Since this 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. Furthermore, to better cover the full space of callable units and binding arguments, we used the same Amazon Mechanical Turk procedure detailed above to collect an additional 352 goal-oriented commands. This resulted in the addition of 4 unique callable unit/binding argument pairs not originally present in the dataset used for the efficiency evaluation.

To produce a dataset of action-specifying callable units, experts annotated low-level trajectory specifications from the efficient execution 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 spaces," "finally left four spaces," 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 1 with their corresponding callable unit and binding arguments. Our final dataset after the additional round of data collection and segmentation consists of 4086 natural language commands total.

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. The second is a **zero-shot** dataset, which consists of a specific train-test split that evaluates how well models can predict

	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%**

(a) IBM2 Reward Grounding Baselines

	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%**

(b) Single-RNN Reward Grounding Baselines

Figure 5: Task grounding accuracy (averaged over 5 trials) when training IBM2 and Single-RNN models on a single level of abstraction, then evaluating commands from alternate levels. This is similar to the MacGlashan et al. [2015] results, as we see that without accounting for abstractions in language, there is a noticeable effect on grounding accuracy.

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

previously unseen action sequence and goal combinations. For example, in this dataset the action-oriented 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. The goal-oriented "zero-shot" dataset was constructed similarly. In both datasets, we assume the test environment is configured the same as the train environment.

6.3 Abstraction & Efficiency

Before diving into the efficiency of our grounding models that simultaneously infer reward function and level of abstraction, we verify that the models can achieve high grounding accuracy. We then conclude with a series of experiments that compare the execution times of the grounded tasks with traditional, flat planning approaches against a hierarchical planner that can leverage the additional abstraction information inferred by our grounding models.

6.3.1 Task Grounding

We present the baseline task grounding accuracies in Figure 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 (for instance, 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 Section 5.1 are shown in Figure 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 (for example, 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 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 Figure 5b; 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.

6.3.2 Robot Execution

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.

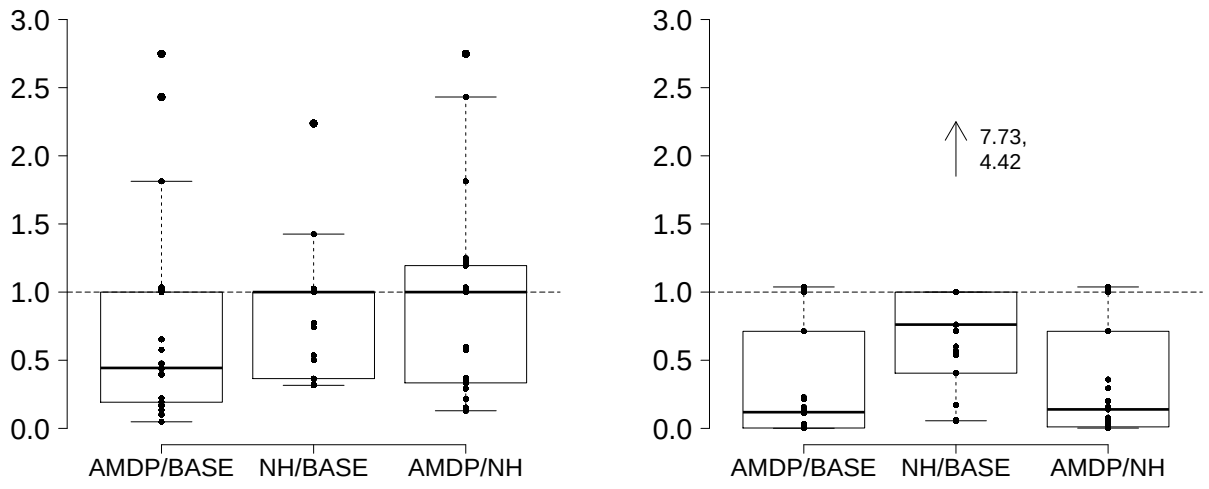
We compared three different planners to solve the MDP:

- **BASE**: A state-of-the-art flat (non-hierarchical) planner, bounded real-time dynamic programming (BRTDP [McMahan et al., 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.
- **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. Figure 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 demonstrating that **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/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



(a) Regular domain (2^{14} states)

(b) Large domain (2^{18} states)

Figure 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.

are agnostic to the discretization of the world. The results are shown in Figure 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.

6.4 Generalization

Unlike efficiency, the ability for a model to generalize beyond what was seen at training time can be measured purely in terms of grounding accuracy. Accordingly, we present language grounding accuracies for our two DRAGGN models in in Table 2 and treat Single-RNN as a baseline.

All three models received the same set of training data, consisting of 2660 low-level action-oriented segments and 1017 high-level goal-based sentences. All together, there are 17 unique combinations of action-oriented callable units and respective binding arguments, alongside 10 unique combinations of goal-oriented callable units and binding arguments present in the data. All three models also received the same set of held-out data consisting of 295 low-level action-oriented segments and 114 high-level goal-oriented 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 now consider these performance numbers by dataset.

	Standard		Zero-Shot	
	Action-Oriented	*Goal-Oriented*	*Action-Oriented*	*Goal-Oriented*
Single-RNN	$95.7 \pm 0.1\%$	$\mathit{84.4 \pm 0.3\%}$	$0.0 + 0\%$	$0.0 \pm 0\%$
J-DRAGGN	$96.3 \pm 0.3\%$	$\mathit{84.5 \pm 0.4\%}$	$21.1 \pm 15.5\%$	$2.4 \pm 1.8\%$
I-DRAGGN	$\mathbf{97.1 \pm 0.3\%}$	$81.0 \pm 0.9\%$	$\mathbf{97.0 + 0.0\%}$	$\mathbf{19.1 \pm 12.6\%}$

Table 2: Accuracy results (mean and standard deviation across 3 random initializations) on both the standard and zero-shot datasets. **Bold** indicates the singular model that performed the best, whereas *italics* denotes the best models that were within the margin of error of each other.

6.4.1 Standard Dataset

Action Prediction: An action-oriented 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). All models perform generally well on this task, with Single-RNN correctly identifying the correct action callable unit on 95.7% of test samples, while both DRAGGN models slightly outperform with on 96.3% and 97.1% respectively.

Goal Prediction: An goal-based command 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 81.0% or higher, with the Single-RNN and J-DRAGGN models being too close to call.

6.4.2 Zero-Shot Dataset

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 ability to disambiguate language pertaining to callable unit prediction from language pertaining to binding argument prediction. This ability to disambiguate between the two outputs seemingly comes from the separate embedding spaces maintained for callable units and binding constraints respectively.

Goal Prediction: The Single-RNN baseline is again completely unable to produce unit-argument pairs never seen during training. However, while both DRAGGN models exhibit non-zero performance, the mean performance is rather low, whereas the variance is quite high. Even with the separate embedding spaces for callable units and binding arguments, the I-DRAGGN is unable to attain the same level of accuracy exhibited in the zero-shot action prediction experiments.

6.5 Robot Demonstrations

We now outline a set of demonstrations conducted on a mobile robot, utilizing the Single-RNN and I-DRAGGN models to ground and execute spoken natural language instructions in real time.

6.5.1 Abstraction

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, 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

6.5.2 Generalization

In this section we use I-DRAGGN to specify goal and action behaviors to a Turtlebot agent in a real environment version of the Cleanup Domain. The natural language verbal command is first converted to text using Google's Speech API [Google, 2017]. Next, we use I-DRAGGN to ground the natural language command to an executable task on the robot. The agent then plans for the tasks using an Abstract Markov Decision Processes hierarchy [Gopalan et al., 2017] to speed up planning on the continuous domain.

The demo first demonstrates the robot's capability to execute action specifications such as "go three steps down" and "go up." Further, we demonstrate the goal based tasks like "go to the green room" and "take the block to the blue room." The robot can ground and plan all these commands real time. We then change the map of the Cleanup domain to show that the robot can still ground the tasks in the new environment given task specifications from I-DRAGGN. A video demonstration of the end to end task is submitted as supplemental material, and is additionally available at this address: https://youtu.be/_u8msi-nZTI.

7 Discussion

We now discuss the experimental results for efficient abstraction and generalization separately.

7.1 Abstraction

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 Figure 4a 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 Figure 4a 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 such as 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 et al. [2006].

7.2 Generalization

Our experiments show that the DRAGGN models improve over existing state-of-the-art in grounding action-oriented language in an object manipulation domain. Furthermore, due to the factored nature of the output, the I-DRAGGN model performs strongly in zero-shot action-oriented prediction.

Nevertheless, I-DRAGGN did not perform as well as Single-RNN and J-DRAGGN on goal-oriented language in the standard setting. 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 shared parameters in each model), 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 in the zero-shot setting, 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 (specifically, 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.

On goal-oriented language in the zero-shot setting, all models exhibit weak performance, although the I-DRAGGN does show some ability to generalize. This seems to indicate the difficulty of zero-shot language grounding in a setting where the instructional language is complex, with multiple overlapping words and phrases, and high variance in expressive power. To better illustrate this, consider two commands from our goal-oriented dataset: "Go to the red room," vs. "Go from the red room to the blue room." While both instructions ground to separate outputs (i.e roomIsRed vs roomIsBlue), there is a lot of overlap in language, which would make it hard for the I-DRAGGN to learn effective embeddings in the binding argument space. Because goal-oriented language is less constrained and generally more expressive than action-oriented language, zero-shot prediction is that much more difficult, and remains an open problem.

While our results demonstrate that the DRAGGN framework is effective, more experimentation is needed to fully explore the possibilities and weaknesses of such models. While we've already identified zero-shot goal prediction as one such limitation, another shortcoming 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.

8 Conclusion

Real-world language grounding systems must extend beyond highly accurate identification of tasks specified through natural language if they are to become widespread throughout the home and workplace. These systems must not only be accurate in their interpretation of language but also be efficient in their execution of the identified task and be capable of leveraging linguistic structure to generalize beyond commands only seen at training time. In this work, we show that the choice of semantic goal representation used within a language grounding model has powerful implications for its ability to meet these three desired criteria. Moreover, we introduce two distinct grounding models, each with their own unique goal representation, that individually demonstrate improved efficiency through abstraction and generalization over baseline models. By explicitly considering the latent level of abstraction in language, our Single-RNN model can interpret a much wider range of natural language commands and leverage an existing hierarchical planner for efficient execution of robot tasks. In parallel, our DRAGGN architectures factor the output space according to the compositional structure of our semantic representation thereby allowing the model to exhibit mild generalization to commands not seen during training.

There are limitations of this work that offer rich directions for future language grounding research. Here we have assumed that all language commands exist at precisely one level of abstraction; clearly, in natural human discourse, people shift freely between levels of abstraction within a single instruction. For instance, "go to the kitchen and take a few steps to your left." Additionally, humans naturally use referential expressions in language to quickly resolve ambiguities. Through the use of determiners, we would enable future language grounding systems to handle commands like "bring the chair that's in the blue room." Commands that not only indicate what should be done but also specify how it should be done pose another challenge. Allowing for commands to be qualified by adverbs like "quickly" or "safely" could require another extension of a semantic representation to explicitly model the qualifiers. In parallel, our factored output space, while being representative of a large body of tasks, is still relatively small. We plan on scaling up to domains with larger output spaces potentially requiring us to explore other sequence-to-sequence mapping approaches that can better handle the increase.

9 Acknowledgements

This work is supported by the National Science Foundation under grant number IIS-1637614, the US Army/DARPA under grant number W911NF-15-1-0503, and the National Aeronautics and Space Administration under grant number NNX16AR61G.

Lawson L.S. Wong was supported by a Croucher Foundation Fellowship.

References

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

Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. In *Robotics: Science and Systems*, 2017.

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2000.

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

Peter F. Brown, Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311, 1993.

David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI Conference on Artificial Intelligence*, 2011.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gúlçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*, 2014.

Junyoung Chung, Çaglar Gúlçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Neural Information Processing Systems Workshop on Deep Learning*, 2014.

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

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

Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul Schermerhorn. 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*, 2009.

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

Nakul Gopalan, Marie desJardins, Michael L. Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, Robert John Winder, and Lawson L. S. Wong. Planning with abstract Markov decision processes. In *International Conference on Automated Planning and Scheduling*, 2017.

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

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

Mohit Iyyer, Varun Manjunatha, Jordan L. Boyd-Graber, and Hal Daumé. Deep unordered composition rivals syntactic methods for text classification. In *Annual Meeting of the Association for Computational Linguistics*, 2015.

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

Andreas Junghanns and Jonathan Schaeeer. 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*, 1997.

Siddharth Karamcheti, Edward C. Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex. A tale of two DRAGGNs: A hybrid approach for interpreting action-oriented and goal-oriented instructions. In *Annual Meeting of the Association for Computational Linguistics Workshop on Language Grounding for Robotics*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Percy Liang. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76, 2016.

James MacGlashan, Monica Babeş-Vroman, Marie DesJardins, Michael L. Littman, Smaranda Muresan, Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding English commands to reward functions. In *Robotics: Science and Systems*, 2015.

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

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

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

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

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2011.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M. Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. In *Robotics: Science and Systems*, 2016.

Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. ROS: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, 2009.

Scott E. Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations*, 2016.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems*, 2014.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

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

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

Tatsuro Yamada, Shingo Murata, Hiroaki Arie, and Tetsuya Ogata. Dynamical linking of positive and negative sentences to goal-oriented robot behavior by hierarchical RNN. In *International Conference on Artificial Neural Networks*, 2016.

John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *National Conference on Artificial Intelligence*, 1996.

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence*, 2005.