# Accurately and Efficiently Interpreting Human-Robot Instructions of Varying Granularities

Dilip Arumugam*, Siddharth Karamcheti*, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex
Computer Science Department, Brown University, Providence, RI 02912
{dilip_arumugam@, siddharth_karamcheti@, ngopalan@cs., lsw@, stefie10@cs.}brown.edu
* The first two authors contributed equally



***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, those approaches that do not use abstraction experience inefficient planning and execution times due to the large, intractable state-action spaces, which closely resemble 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.**

## I. 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 forklift 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 [6, 19, 30]. While effective at directing robots to complete predefined tasks, mapping to fixed sequences of robot actions is unreliable when faced with a changing or stochastic environment. Accordingly, MacGlashan et al. [17] 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. 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 [11].

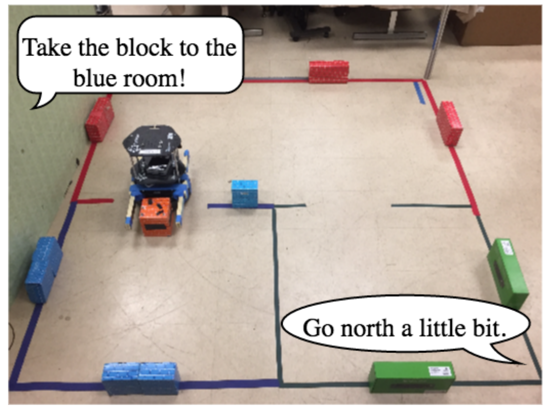



Fig. 1: Examples of high-level and fine-grained commands issued to the Turtlebot robot in a mobile-manipulation task.

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 learns how to map natural language commands to the appropriate level of the planning hierarchy. By coupling abstraction level inference with the overall grounding problem, we fully 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 the deep neural network language model can infer reward functions faster and 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.

## II. Related work

Humans use natural language to communicate ideas, motivations, task descriptions, etc. with other humans. One of the earliest works in this area mapped tasks to another planning language, which then grounded to the actions performed by the robots [6]. More recent methods ground natural language commands to tasks using features that describe correspondences between natural language phrases present in the task description and physical objects and actions available in the world [12, 19, 30]. This featurized representation can then describe the sequence of actions needed to complete the task. In a similar vein Paul et al. [25] ground to abstract spatial concepts like *rows*, *columns* and *middle* before learning correspondences between them to solve tasks. All these approaches ground commands to action sequences, leading to brittle behavior if the environment is stochastic.

MacGlashan et al. [17] proposed grounding natural language commands to reward functions associated with certain tasks, allowing robot agents to plan in stochastic environments. The robot can solve for individual plans once natural language commands ground to applicable reward functions. MacGlashan et al. [17] treated 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 [4, 5] (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.

Planning in domains with large state-action spaces is computationally expensive as planners like value iteration and bounded RTDP need to explore the domain at the lowest, "flat" level of abstraction [2, 21]. 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 [20] or Options [29]. These methods achieve subgoals using either a fixed sequence of actions [20] or a subgoal based policy [29]. 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 [9], R-MAXQ [14] and Abstract Markov Decision Processes (AMDPs) [11] 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 [9, 11, 14]. Both MAXQ [9] and R-MAXQ [14] are bottom-up planners, they back up each individual action's reward across the hierarchy.

We chose AMDPs [11] for our approach because they plan in a "top-down" fashion. Much like in an MDP, 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 [11]; the states of the subtask AMDP are abstract representations of the environment state. AMDPs have been shown to achieve faster planning performance than other hierarchical methods [11]

To perform the language grounding, we use a deep neural network language model. Deep neural networks have had great success in a variety of natural language tasks, like traditional language modeling [3, 22, 23], machine translation [7, 8], and text categorization [13]. One of the contributing factors to the success of such methods is their ability to learn meaningful input representations during the training process. For example, Bengio et al. [3] learn distributed representations of words in tandem with the rest of their language model. Similarly, Mikolov et al. [24] propose a system solely dedicated to learning these representations. These "embeddings" are dense vectors that not only uniquely represent individual words (as opposed to otherwise sparse approaches for word representation), but capture semantically significant features of the language as well. Another contributing factor to the success of such methods is the use of Recurrent Neural Networks, 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 [7, 8]. To the best of our knowledge, our approach is the first to use both word embeddings and a state-of-the-art RNN model to map between natural language and an MDP reward function.

## III. Technical Approach

To interpret a variety of natural language commands, there must be a representation for all possible tasks and subtasks. We define an *Object-oriented Markov Decision Process* (OO-MDP) to represent the robot's actions [10]. 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 $\gamma$ 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. [17], 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. 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 [15, 17] OO-MDP, which models a mobile manipulator robot; this domain is defined in Sec. V-A.

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 [11]. 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 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 reward function $\hat{m}$ such that:

$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(l, m \mid c) \tag{1}$$

For example, a high-level natural language command like "Take the block to the blue room" (as shown at the top of Fig. 1) would map to the highest level of abstraction while a low-level command like "Go north a little bit" (as shown at the bottom of Fig. 1) would map to the finest-grained level of abstraction. We estimate this joint probability by learning a language model (described in Sec. IV) 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. 7. Specifically, we learn a set of parameters $\hat{\theta}$ that maximize the following corpus likelihood:

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

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

## IV. Language Models

We compare four language models: an IBM Model 2 translation model (similar to MacGlashan et al. [17]), a deep neural network bag-of-words language model, and two sets of recurrent neural network language models, with varying architectures. For detailed descriptions of all the presented models, please refer to the supplementary material.

### A. IBM Model 2

As a baseline, task grounding is recast as a machine translation problem with a source language defined by natural language and a target language defined by semantic task representations (reward functions). We use the well known IBM Model 2 (IBM2) machine translation model [4, 5] as a statistical language model for scoring reward functions based on some input command. IBM2 is a generative model that solves the following objective, which is equivalent to Eqn. 7 by Bayes' rule:

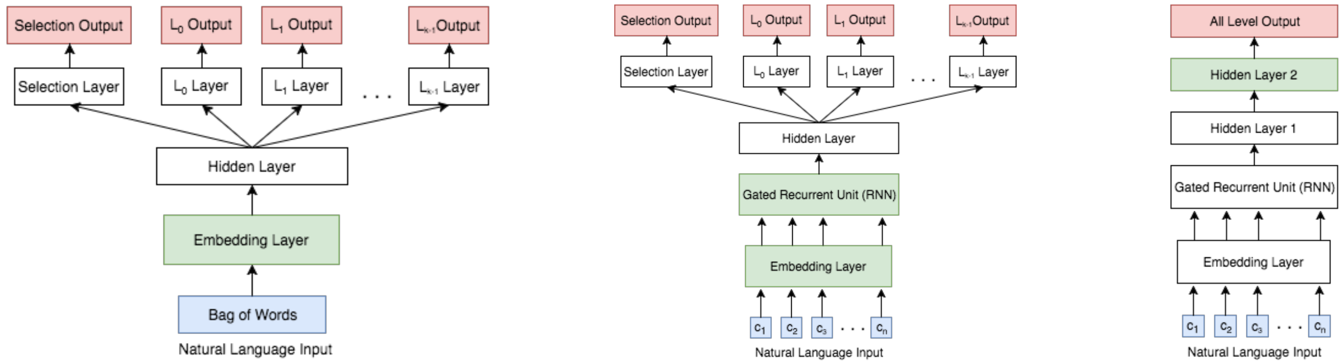
$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(l, m) \cdot Pr(c \mid l, m) \tag{3}$$

This task grounding formulation follows directly from MacGlashan et al. [17] and we continue in an identical fashion training the IBM2 using the standard EM algorithm.

### B. Neural Network Language Models

We evaluate three classes of neural network architectures (see Fig. 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**).

*1) **Multi-NN: Multiple Output Feed-Forward Network**:* We propose a feed-forward neural network [3, 13, 24] that takes in a natural language command as a bag-of-words, and outputs both the probability of each of the different levels of abstraction, as well as the probability of each reward function. Specifically, we decompose the conditional probability from Eqn. 7 as $Pr(l, m \mid c) = Pr(l \mid c) \cdot Pr(m \mid l, c)$. If we represent our natural language command $c$ as a bag-of-words




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

Fig. 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.

vector $\vec{c}$, the Multi-NN objective is to find a set of parameters $\hat{\theta}$ such that:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(\vec{c},l,m)} \log Pr(l \mid \vec{c}, \theta) + \log Pr(m \mid l, \vec{c}, \theta) \quad (4)$$

This follows by taking a logarithm of the corpus objective outlined in Eqn. 2.

To learn this set of parameters, we use the architecture shown in Fig. 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. 4. We train the network via backpropagation, using the Adam Optimizer [16], 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 [27] after the initial embedding layer, as well as after the output-specific hidden layers with probability $p = 0.5$.

*2) **Multi-RNN: Multiple Output Recurrent Network**:* Inspired by the success of recurrent neural networks in NLP tasks [7, 22, 23, 28], we propose a recurrent neural network 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 of the different levels of abstraction. Recurrent Neural Networks are extensions of feed-forward networks that can handle variable length inputs. They do this by employing a set of one or more hidden states which update after reading in each input token. Similar to the Multi-NN, we decompose the conditional probability from the objective in Eqn. 7. If we represent the natural language command $c$ as a sequence of words $s = \langle c_1, c_2 \dots c_n \rangle$, the Multi-RNN objective is to find a set of parameters such that:

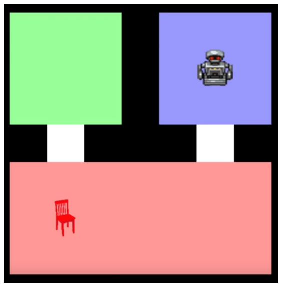
$$\hat{\theta} = \arg\max_{\theta} \sum_{(c,l,m)} \log Pr(l \mid s, \theta) + \log Pr(m \mid l, s, \theta) \quad (5)$$

To do this, we use the architecture depicted in Fig. 2b; similar to the Multi-NN architecture, we instead use a Recurrent Neural Network encoder that takes the sequence of raw input tokens (in lieu of a bag-of-words representation), and maps them into a fixed-size state vector.

In this work, we leverage the the Gated Recurrent Unit (GRU) of Cho et al. [7], a particular type of Recurrent Neural Network cell that is characterized by a hidden state incrementally updated with new inputs (i.e. words in a command). We utilize them specifically as they have been shown to work well on natural language sequence modeling tasks [8]. Similar to the Multi-NN, we train the model by minimizing the sum of the cross-entropy loss of each of the two terms described in objective Eqn. 5, with the same optimizer setup as the Multi-NN model. Once again, dropout is used to regularize the network, after the initial embedding layer, as well as after the output-specific hidden layers.

*3) **Single-RNN: Single Output Recurrent Network**:* Both the Multi-NN and the Multi-RNN approaches detailed above decompose the conditional probability of both the level of abstraction $l$ and the 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. Furthermore, as a result of this decomposition, both the Multi-NN and Multi-RNN create separate sets of parameters for each of the separate outputs - namely, this translates to separate sets of parameters for each of the levels of abstractions in the underlying hierarchical planner.

Alternatively, it is possible to use a model to directly estimate the joint probability $Pr(l, m \mid c)$. To do this, we propose a different type of recurrent neural network model that takes in a natural language command as a sequence of words, and directly outputs the joint probability of each tuple $(l, m)$, where $l$ denotes the level of abstraction, and $m$ denotes



(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 room1 |
| $L_1$ | Go to door, enter red room, push chair to green room door. | blockInRegion block0 room1 |
| | Go to the door then go into the red room. | agentInRegion agent0 room0 |
| $L_2$ | Go to the green room. | agentInRegion agent0 room1 |
| | Bring the chair to the blue room. | blockInRegion block0 room2 |

(b) Example commands and corresponding reward functions.

Fig. 3: Amazon Mechanical Turk (AMT) dataset statistics and examples.

the reward function at the given level. Specifically, if we represent the natural language command $c$ as a sequence of words $s = \langle c_1, c_2 \ldots c_n \rangle$, then the Single-RNN objective is to find a set of parameters $\hat{\theta}$ such that:

$$\hat{\theta} = \arg\max_{\theta} \sum_{(n,l,m)} \log Pr(l, m \mid s, \theta) \tag{6}$$

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.

With this in mind, we use the architecture depicted in Fig. 2c. Namely, we employ a single-output recurrent neural network, similar to the Multi-RNN architecture, with the key difference 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 given the natural language command.

To train the Single-RNN model, we directly minimize the cross-entropy loss of the joint probability term described in objective Eqn. 6. Training hyperparameters are identical to the Multi-RNN model and dropout is applied to the initial embedding layer as well as the penultimate hidden layer.

## V. Evaluation

The aim of our evaluation is to test the hypothesis that hierarchical structure improves the speed and accuracy of language grounding at multiple levels of abstraction. We evaluate our method with a corpus-based evaluation in simulation and assess the speed and accuracy of our approach. Additionally we demonstrate our system on a Turtlebot mobile robot.

### A. Mobile Robot Domain

The Cleanup World is a mobile-manipulator robot domain that is partitioned into rooms (denoted by unique colors) with open doors. Additionally, 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 must move objects around an environment, analogous to a robotic forklift operating in a warehouse or a pick-and-place robot in a home environment. We use an AMDP taken directly from Gopalan et al. [11] for the Cleanup World domain which imposes a three-level abstraction hierarchy that can be utilized for planning.

The combinatorially large state space of the Cleanup World simulates real world complexity and is ideal for exploiting abstractions. At the lowest level of abstraction (which we refer to as $L_0$), the (primitive) action set available to the robot agent consists of north, south, east, and west actions. Accordingly, a user directing the robot at this level of granularity must specify lengthy commands representing step by step instructions for the robot to execute. At the next level of abstraction (referred to as $L_1$), the topology of the Cleanup World is drastically reduced to only consist of rooms and doors. The robot's presence in the domain is solely defined by the region (*i.e.* room or door) it resides in; the abstracted actions here denote 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 (*i.e.* it would not be possible to directly move from the green room to the blue room in Fig. 3a without first passing through the red room); any language guiding the robot at this level must adhere to these dynamics. Finally, the last level of abstraction (referred to as $L_2$) mimics the previous but removes the concept of doors leaving only whole rooms as regions; all transition dynamics still hold including adjacency as deciding whether or not the robot may move to another room from it's current room. Here, subroutines exist for moving either the robot or an available object between connected rooms. The full space of subroutines at all levels and their corresponding propositional functions are defined by [11] however Fig. 3b provides a few collected sample commands at each level and the corresponding level-specific subroutine within the AMDP.

### B. Procedure

Given the AMDP for the Cleanup World domain we conducted an Amazon Mechanical Turk (AMT) user study to collect natural language samples at various levels of abstraction for the Cleanup World domain [15, 17] (see Fig. 3a). Annotators were shown video demonstrations of ten tasks

| | 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

Fig. 4: 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. [17] 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**% |

Fig. 5: Accuracy of 10-Fold Cross Validation (averaged over 3 runs) for each of the models on the AMT Dataset.

using a single starting instance of the Cleanup World domain shown in Fig. 3a. For each task, we asked them to type a command that they would to ask 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 was used to construct multiple 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. [17], 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"). In order to not limit the potential variation in the provided language commands, only those commands which reflected a clear misunderstanding of the presented task were removed from the dataset. At the end of data collection, we removed fewer than 30 commands for this reason giving us a total of 3047 commands; an example removed command was the vague fragment, "please robot" as it is not clear from this example what the robot should be doing. Per level, this gave us 1309 commands at $L_0$, 872 commands at $L_1$, and 866 commands at $L_2$. 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 or exist at higher levels of abstraction.

### C. Robot Task Grounding

In order to demonstrate the importance of inferring the latent abstraction level in language, we present the baseline task grounding accuracies in Fig. 4. 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 CleanupWorld hierarchy were omitted from the dataset, 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.* $L_1$ 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. [17] 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, the Single-RNN model shows the expected maximization along diagonal entries that comes from training and evaluating on data at the same level of abstraction. These show 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, the Single-RNN model demonstrates more robust task grounding than statistical machine translation.

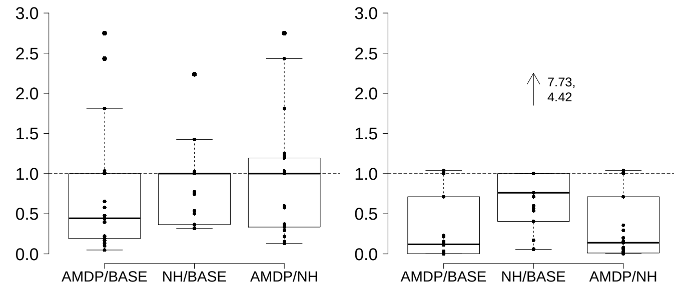
The task grounding and level inference scores for the models in Sec. IV are shown in Fig. 5. Attempting to embed the latent abstraction level within the machine language of IBM2 results in weak level inference. Furthermore, grounding accuracy falls even further due to sparse alignments and the sharing 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. The Multi-RNN model 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, the Single-RNN model 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. 4b; 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.

### D. 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. To assess response time, 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 [21]).
- **AMDP**: A hierarchical planner for MDPs [11]. 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)

Fig. 6: 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. 6a shows the results for all 3 pairs of planners. For example, the left-most column shows $\frac{\textbf{AMDP}\text{ time}}{\textbf{BASE}\text{ time}}$; 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/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. 6b, 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 [9]. 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 $< 1$s on $90\%$ of the tasks; in contrast, **BASE** takes $> 20$s on half the tasks.

### *E. Robot Demonstration*

Using the trained grounding model and the corresponding AMDP hierarchy, we tested on a Turtlebot within a small scale version of the Cleanup World domain. In order 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. These commands were implemented using low level, closed loop control policies, which were sent to the robot using the Robot Operating System [26].

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 [1] 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 system running end to end is available online [1].

## VI. Discussion

We present baseline results highlighting the difficulty of task grounding when modeling nuance in natural language, and go on to show dramatic improvement using our proposed methods. Furthermore, we show that while ignoring the underlying structure of natural language may result in better task grounding performance, the remaining planning problem cannot be solved efficiently for large domains. Together, our results demonstrate that we can maintain highly accurate task grounding as well as robust, efficient planning in complex environments. Finally, we demonstrate an end-to-end system using our approach deployed on a mobile robot.

[1] https://youtu.be/9bU2oE5RtvU

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. 3a is "Go down...west until you hit the chair, push chair north..." Alternatively, 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. 3a 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 at a different level of abstraction then 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. [18].

## VII. Conclusion

In this paper, we presented a system for interpreting and grounding natural language commands to a mobile-manipulator (pick-and-place) robot at multiple levels of abstraction. To the best of our knowledge, our system is not only the first work to ground language at multiple levels of abstraction, but also the first to utilize deep neural networks for language grounding on robots. We demonstrate that integrating such a language grounding system with a hierarchical planner allows for the specification and efficient execution of a wide range of robot tasks, fostering a very natural human to robot interaction. Additionally, through our Turtlebot demonstrations, we show that this system works well in real-world environments.

Future work should extend this system via application to a large variety of real-world scenarios. Such a system would be effective in any environment where having multiple levels of abstraction make sense; for example, in surgical and household robotics. Additionally, it would be incredibly fruitful to extend the models proposed here to operate on natural language commands specified at a mixture of abstraction levels to further reduce the constraints on natural language and facilitate a more natural human-robot interaction. Alternate future work would relax the assumptions made in this work and allow for full variation in language or full variation in planning abstraction; one might, for example, learn abstraction hierarchies directly from the language.

## VIII. 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

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

[2] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] Junyoung Chung, Çaglar Gúlçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

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

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

[11] Nakul Gopalan, Marie desJardins, Michael L. Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, and Lawson L.S. Wong. Planning with abstract markov decision processes. In *International Conference on Machine Learning Workshop on Abstraction in Reinforcement Learning*, 2016.

[12] 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.

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

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

[15] 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.

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

[17] 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.

[18] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *National Conference on Artificial Intelligence*, volume 2, page 4, 2006.

[19] 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.

[20] 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.

[21] H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and per-

formance guarantees. In *International Conference on Machine Learning*, 2005.

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

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[29] 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.

[30] 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.

## Appendix

Here we provide details of the various language modeling approaches used for task grounding including a breakdown of the specific details of the IBM Model 2 Language Model, as well as each of the separate deep neural network models.

Recall that, given a natural language command $c$, we find the corresponding level of the abstraction hierarchy $l$, and the 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 reward function $\hat{m}$ such that:

$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(l, m \mid c) \tag{7}$$

### A. IBM Model 2 - Statistical Language Model:

IBM2 is a generative model that solves the following objective, which is equivalent to Eqn. 7 by Bayes' rule:

$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(l, m) \cdot Pr(c \mid l, m) \tag{8}$$

In this equation, the first term, $Pr(l, m)$ can be treated as a distribution over the reward function space. We make the assumption that each $(l, m)$ tuple is distributed uniformly at random. Thus, the IBM2 learning objective simplifies to the following:

$$\hat{l}, \hat{m} = \arg\max_{l,m} Pr(c \mid l, m) \tag{9}$$

This probability of a natural language command ($c$) given the reward ($m$) and level of abstraction ($l$) is then given by the following IBM2 equation:

$$Pr(c|m, l) = \eta(n_c|n_m, l) \sum_a \prod_j^{n_c} \delta(a_j|j, n_c, n_m, l)\tau(c_j|m_{a_j}, l) \tag{10}$$

where $\eta$, $\delta$, and $\tau$ are IBM2 specific parameters that are learned via the EM algorithm. $\eta(n_c \mid n_m, l)$ denotes the probability of generating a natural language command of length $n_c$ from a reward function of length $n_m$, and level $l$. The sum is defined over all possible alignments of natural language words to reward function tokens. For computational efficiency, we approximate the sum by sampling from the set of possible alignments, following standard practice.

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

At inference time, to pick the $(l, m)$ tuple that maximizes the objective from Equation 9 we calculate the IBM2 probability for every possible $(l, m)$ combination, using the IBM2 as a reranker 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.

### B. Multi-NN - Multiple Output Feed-Forward Network:

A breakdown of the exact network transformations is as follows:

$$\begin{aligned}
\vec{e} &= \text{Lookup}(\mathbf{E}, \vec{c}) \\
\vec{s} &= \text{ReLU}(\vec{e} \cdot \mathbf{W_s} + \mathbf{b}_s) \\
\vec{t} &= \text{ReLU}(\vec{s} \cdot \mathbf{W_t^k} + \mathbf{b}_t^k) \\
\vec{o} &= \text{Softmax}(\vec{t} \cdot \mathbf{W_t^k} + \mathbf{b}_t^k)
\end{aligned}$$

Here, the layer specific weight and bias parameters are given by $\mathbf{W}, \mathbf{b}$ respectively. Superscripts denote output-specific parameters and the $(\cdot)$ operation denotes matrix-vector product. In order to produce high-dimensional, fixed-size representations of each word in the finite natural language vocabulary, the initial embedding layer contains a lookup matrix $\mathbf{E}$, trained via backpropagation with the rest of the model, where each row denotes a single word embedding. The embedding for all words in $\vec{c}$ are summed together according to their respective frequencies to produce an embedding for the full natural language command. All hidden layers employ the rectifier activation function (ReLU) whereas the final output layer produces a Softmax distribution over the output categories.

The fixed-size embedding is then passed through a neural network layer (shared across all outputs), with a ReLU non-linear activation, generating a hidden state vector $\vec{h}$. This hidden state vector is passed through an output-specific hidden layer, also with a ReLU activation, and finally an output-specific read-out layer, with a Softmax activation, to generate a probability distribution over the output categories. The loss is computed as the sum of the cross-entropy loss over the different outputs - namely, the computed loss for the level selection distribution, and each of the three different reward function distributions.

### C. Gated Recurrent Units

Both the Multi-RNN and Single-RNN models leverage Gated Recurrent Unit (GRU) cells, a specific type of Recurrent Neural Network cell. GRU Cells only maintain a single hidden state $h$, and the update rules are as follows:

$$\begin{aligned}
\vec{z}_t &= \sigma(\mathbf{W_z} \cdot \vec{x}_t + \mathbf{U_z} \cdot \vec{h}_{t-1} + \mathbf{b}_z) \\
\vec{r}_t &= \sigma(\mathbf{W_r} \cdot \vec{x}_t + \mathbf{U_r} \cdot \vec{h}_{t-1} + \mathbf{b}_r) \\
\vec{n}_t &= \text{Tanh}(\mathbf{W_h} \cdot \vec{x}_t + \mathbf{U_h} \cdot (\vec{r}_t \odot \vec{h}_{t-1}) + \mathbf{b}_z) \\
\vec{h}_t &= (\vec{1} - \vec{z}_t) \odot \vec{h}_{t-1} + \vec{z}_t \odot \vec{n}_t
\end{aligned}$$

Here, the $(\cdot)$ operation denotes matrix-vector product, while the $(\odot)$ operation denotes element-wise product. The intermediate vectors $\vec{z}, \vec{r}$ act as update and reset "gates" dictating how much of the hidden state should be overwritten with the new information in $x_t$. The parameters $\mathbf{W}, \mathbf{U}, \mathbf{b}$ are specific to the GRU cell, and are trainable via backpropagation along with the rest of the model. The hidden state $h$ is initialized as the zero vector at $t = 0$.

### D. *Multi-RNN - Multiple Output Recurrent Network:*

We now give a detailed breakdown of the exact network transformations that make up the Multi-RNN:

$$\begin{aligned}
\vec{e}_1, \vec{e}_2 \ldots \vec{e}_n &= \text{Lookup}(\mathbf{E}, c_1, c_2 \ldots c_n) \\
\vec{h} &= \text{GRU}(\vec{e}_1, \vec{e}_2, \ldots \vec{e}_n) \\
\vec{s} &= \text{ReLU}(\vec{h} \cdot \mathbf{W_s} + \mathbf{b}_s) \\
\vec{t} &= \text{ReLU}(\vec{s} \cdot \mathbf{W_t^k} + \mathbf{b}_t^k) \\
\vec{o} &= \text{Softmax}(\vec{t} \cdot \mathbf{W_t^k} + \mathbf{b}_t^k)
\end{aligned}$$

Again, layer parameters are given by $\mathbf{W}, \mathbf{b}$, with superscripts denoting output-specific parameters. The loss is the same as that used by the Multi-NN model.

### E. *Single-RNN - Single Output Recurrent Network:*

A detailed breakdown of the Single-RNN transformations are as follows:

$$\begin{aligned}
\vec{e}_1, \vec{e}_2 \ldots \vec{e}_n &= \text{Lookup}(\mathbf{E}, c_1, c_2 \ldots c_n) \\
\vec{h} &= \text{GRU}(\vec{e}_1, \vec{e}_2, \ldots \vec{e}_n) \\
\vec{s} &= \text{ReLU}(\vec{h} \cdot \mathbf{W_s} + \mathbf{b}_s) \\
\vec{t} &= \text{ReLU}(\vec{s} \cdot \mathbf{W_t} + \mathbf{b}_t) \\
\vec{o} &= \text{Softmax}(\vec{t} \cdot \mathbf{W_t} + \mathbf{b}_t)
\end{aligned}$$

Note that these transformations are exactly the same as the Multi-RNN, with the sole exception that there is only a single output, rather than multiple. As there is only a single output, the new loss is just the cross-entropy loss of the predicted joint level-reward function distribution.