A Tale of Two DRAGGNs: A Hybrid Approach for Interpreting Action-Oriented and Goal-Oriented Instructions

Siddharth Karamcheti, Edward C. Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson L.S. Wong, Stefanie Tellex
Department of Computer Science, Brown University, Providence, RI 02912
{siddharth_karamcheti@, edward_c_williams@, dilip_arumugam@, mina_rhee@, ngopalan@cs., lsw@, stefie10@cs.}brown.edu

Abstract

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

1 Introduction

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

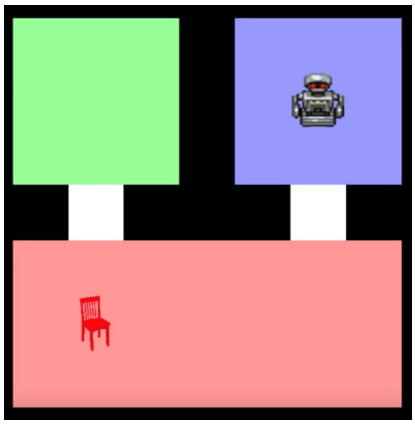


Figure 1: Sample configuration of the Cleanup World mobile-manipulator domain (MacGlashan et al., 2015), used throughout this work. A possible goal-based instruction could be "Take the chair to the green room," while a possible action-based instruction could be "Go three steps south, then two steps west."

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

Existing approaches can be broadly divided into one of two regimes. Goal-based approaches like

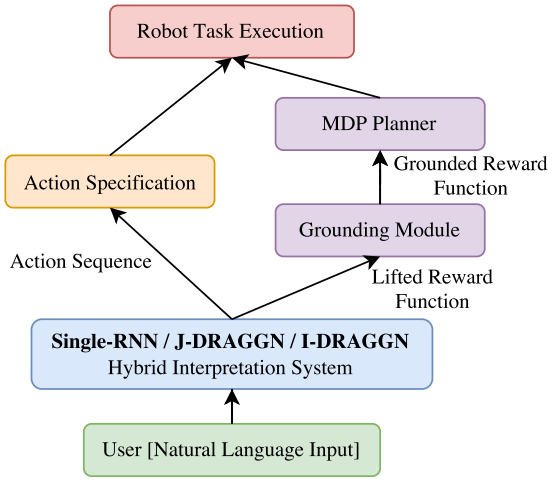


Figure 2: System for grounding both action-oriented (left branch) and goal-oriented (right branch) natural language instructions to executable robot tasks. Our main contribution is the hybrid interpretation system (blue box), for which we present two novel models based on the DRAGGN framework (J-DRAGGN and I-DRAGGN) in Section 4.

MacGlashan et al. (2015) and Arumugam et al. (2017) leverage some intermediate task representation and then automatically find a low-level trajectory to achieve the goal using a planner. Other approaches, in the action-oriented regime, directly infer action sequences (Tellex et al., 2011; Matuszek et al., 2012; Artzi and Zettlemoyer, 2013; Andreas and Klein, 2015) from the syntactic or semantic parse structure of natural language. However, these approaches can be computationally intractable for large state-action spaces or use ad-hoc methods to execute high-level language rather than relying on a planner. Furthermore, these methods are unable to adapt to dynamic changes in the environment; for example, consider an environment in which the wind, or some other force moves an object that a robot has been tasked with picking. Action sequence based approaches would fail to handle this without additional user input, while goal-based approaches would be able to replan on the fly, and complete the task.

To address the issue of dealing with both goal-oriented and action-oriented commands, we present a new language grounding framework that, given a natural language command, is capable of inferring the latent command type. Recent approaches leveraging deep neural networks have formulated the language grounding problem as sequence-to-sequence learning or multi-label classification (Mei et al., 2016; Arumugam et al., 2017). Inspired by the recent success of neural networks to model programs that are highly compositional and sequential in nature, we present the Deep Recurrent Action/Goal Grounding Network (DRAGGN) framework, derived from the the Neural Programmer-Interpreter (NPI) of Reed and de Freitas (2016) and outlined in Section 4.2. We introduce two instances of DRAGGN models, each with slightly different architectures. The first, the Joint-DRAGGN (J-DRAGGN) is defined in Section 4.3, while the second, the Independent-DRAGGN (I-DRAGGN) is defined in Section 4.4.

2 Related Work

There has been a broad and diverse set of work examining how best to interpret and execute natural language instructions on a robot platform (Vogel and Jurafsky, 2010; Tellex et al., 2011; Artzi and Zettlemoyer, 2013; Howard et al., 2014; Andreas and Klein, 2015; Hemachandra et al., 2015; MacGlashan et al., 2015; Paul et al., 2016; Mei et al., 2016; Arumugam et al., 2017). Vogel and Jurafsky (2010) produce policies using language and expert trajectories based rewards, which allow for planning within a stochastic environment along with re-planning in case of failure. (Tellex et al., 2011) instead grounds language to trajectories satisfying the language specification. (Howard et al., 2014) chose to ground language to constraints given to an external planner, which is a much smaller space to perform inference over than trajectories. MacGlashan et al. (2015) formulate language grounding as a machine translation problem, treating propositional logic functions as both a machine language and reward function. Reward functions or cost functions can allow richer descriptions of trajectories than plain constraints, as they can describe preferential paths. Additionally, Arumugam et al. (2017) simplify the problem from one of machine translation to multi-class classification, learning a deep neural network to map arbitrary natural language instructions to the corresponding reward function.

Informing our distinction between action sequences and goal state representation is the division presented by Dzifcak et al. (2009), who posited that natural language can be interpreted as *both* a goal state specification and an action specification. Rather than producing both from

each language command, our DRAGGN framework makes the simplifying assumption that only one representation captures the semantics of the language; additionally, our framework does not require a manually pre-specified grammar.

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

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

3 Problem Setting

We consider the problem of mapping from natural language to robot actions within the context of Markov decision processes. A Markov decision process (MDP) is a five-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ defining a state space $\mathcal{S}$, action space $\mathcal{A}$, state transition probabilities $\mathcal{T}$, reward function $\mathcal{R}$, and discount factor γ (Bellman, 1957; Puterman, 1994). An MDP solver produces a policy that maps from states to actions in order to maximize the total expected discounted reward.

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

We focus our evaluation of all models on the the Cleanup World mobile-manipulator domain (MacGlashan et al., 2015; Arumugam et al., 2017). The Cleanup World domain consists of an agent in a 2-D world with uniquely colored rooms and movable objects. A domain instance is shown in Figure 1. The domain itself is implemented as an object-oriented Markov decision process (OO-MDP) where states are denoted entirely by collections of objects, with each object having its own identifier, type, and set of attributes (Diuk et al., 2008). Domain objects include rooms and interactable objects (*e.g* a chair, basket, etc.) all of which have location and color attributes. Propositional logic functions can be used to identify relevant pieces of an OO-MDP state and their attributes; as in MacGlashan et al. (2015) and Arumugam et al. (2017), we treat these propositional functions as reward functions. In Figure 1, the goal-oriented command "take the chair to the green room" may be represented with the reward function blockInRoom block0 room1, where the blockInRoom propositional function checks if the location attribute of block0 is contained in room1.

4 Approach

We now outline the pipeline that converts natural language input to robot behavior. We begin by first defining the semantic task representation used by our grounding models that comes directly from the OO-MDP propositional functions of the domain. Next, we examine our novel DRAGGN framework for language grounding and, in particular, address the separate paths taken by action-oriented and goal-oriented commands through the system as seen in Figure 2. Finally, we discuss two different

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

Table 1: Set of action-oriented and goal-oriented callable units that can be generated by our DRAGGN models in the Cleanup World domain.

implementations of the DRAGGN framework that make different assumptions about the relationship between tasks and constraints. Specifically, we introduce the Joint-DRAGGN (J-DRAGGN), that assumes a probabilistic dependence between tasks (i.e. goUp) and the corresponding arguments (i.e. 5 steps) based on a natural language instruction, and the Independent-DRAGGN (I-DRAGGN) that treats tasks and arguments as independent given a natural language instruction.

4.1 Semantic Representation

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

Goal-oriented callable units (lifted reward functions) are paired with binding arguments that specify properties of environment entities that must be satisfied in order to achieve the goal. These binding arguments are later resolved by the Grounding Module (see Section 4.5) 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 is shown in Table 1.

4.2 Deep Recurrent Action/Goal Grounding Network (DRAGGN)

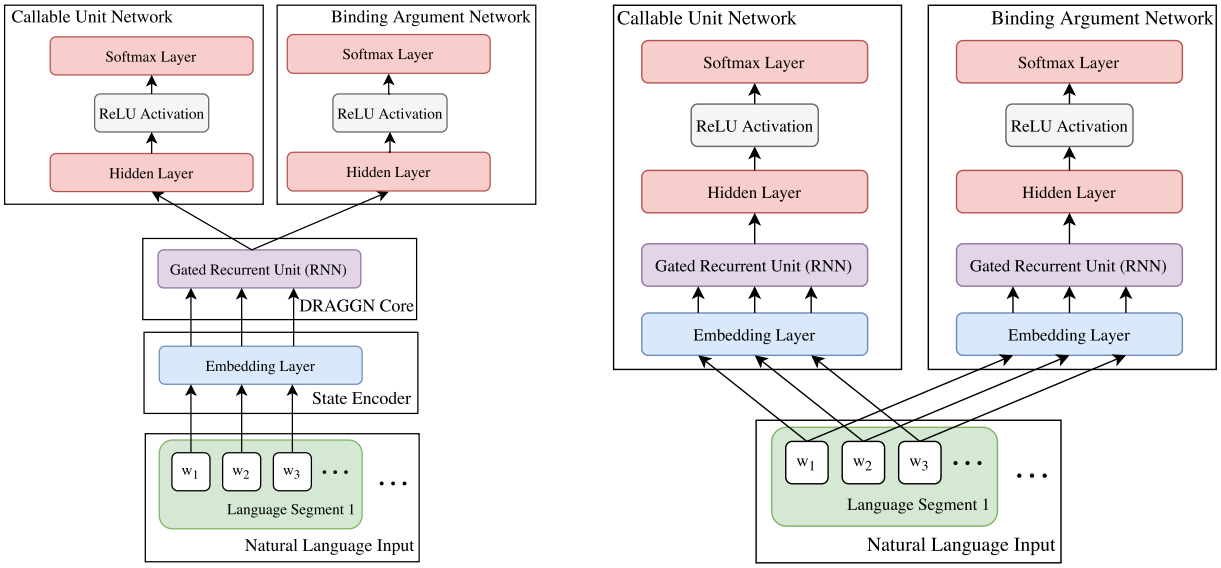
While the Single-RNN model of Arumugam et al. (2017) is effective, it cannot model the compositional argument structure of language. A unit-argument pair not observed at training time will not be predicted from input data, even if the constituent pieces were observed separately. Additionally, the Single-RNN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

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

This unit-argument separation is inspired by the Neural Programmer-Interpreter (NPI) of Reed and 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 Figure 3a, each with its own output space).

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

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



(a) Joint DRAGGN

(b) Independent DRAGGN

Figure 3: Architecture diagrams for the two Deep Recurrent Action/Goal Grounding Network (DRAGGN) models, introduced in Sections 4.3 and 4.4. Both architectures ground arbitrary natural language instructions to callable units (either actions or lifted reward functions), and binding arguments.

maximize the following joint probability:

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

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

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

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

4.3 Joint DRAGGN (J-DRAGGN)

The Joint DRAGGN (J-DRAGGN) models the joint probability in Equation 1, coupled via the shared RNN state in the DRAGGN Core (as depicted in Figure 3a), but selects the optimizer sequentially, as follows:

$$\begin{aligned} \hat{c}, \hat{\mathbf{a}} &= \arg\max_{c,\mathbf{a}} \Pr(c, \mathbf{a} \mid l) \\ &\approx \arg\max_{\mathbf{a}} \left[\arg\max_{c} \Pr(c, \mathbf{a} \mid l) \right] \end{aligned} \tag{2}$$

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[1]. 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

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

vector as input to produce a softmax probability distribution over all possible callable units. The Binding Argument Network is a separate network with an identical architecture and takes the same input, but instead produces a probability distribution over all possible binding arguments. The two models do not need to share the same architecture; for example, callable units with multiple arguments require multiple different argument networks, one for each possible binding constraint.

4.4 Independent DRAGGN (I-DRAGGN)

The Independent DRAGGN (I-DRAGGN), contrary to the Joint DRAGGN, decomposes the objective from Equation 1 by treating callable units and binding arguments as being independent, given the original natural language instruction. More precisely, the I-DRAGGN objective is:

$$\hat{c}, \hat{\mathbf{a}} = \arg\max_{c,\mathbf{a}} \Pr(c \mid l)\ \Pr(\mathbf{a} \mid l) \qquad (3)$$

The I-DRAGGN network architecture is shown in Figure 3b. Beyond the difference in objective functions, there is another key difference between the I-DRAGGN and J-DRAGGN architectures. Rather than encoding the constituent words of the natural language instruction once, and feeding the resulting embeddings through a DRAGGN Core to generate a shared core vector, the I-DRAGGN model embeds and encodes the natural language instruction *twice*, using two separate embedding matrices and GRUs, one each for the callable unit and binding argument. In this way, the I-DRAGGN model encapsulates two disjoint neural networks, each with their own individual parameter sets that are trained independently. The latter half of each individual network (the Callable Unit Network and Binding Argument Network) remains the same as that of the J-DRAGGN.

4.5 Grounding Module

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

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

Table 2: Examples of natural language phrases and corresponding callable units and arguments.

5 Experiments

We assess the effectiveness of both our J-DRAGGN and I-DRAGGN models via instruction grounding accuracy for robot navigation and mobile-manipulation tasks. As a baseline, we compare against the state-of-the-art Single-RNN model introduced by Arumugam et al. (2017).

5.1 Procedure

To conduct our evaluation, we use the dataset of natural language commands for the single instance of Cleanup World domain seen in Figure 1, from Arumugam et al. (2017). In the user study, Amazon Mechanical Turk users were presented with trajectory demonstrations of a robot completing various navigation and object manipulation tasks. Users were prompted to provide natural language commands that they believed would have generated the observed behavior. Since the original dataset was compiled for analyzing the hierarchical nature of language, we were easily able to filter the commands down to only those using high-level goal specifications and low-level trajectory specifications. This resulted in a dataset of 3734 natural language commands total.

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

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

Table 3: Action-oriented and goal-oriented accuracy results (mean and standard deviation across 3 random initializations) on both the standard and unseen datasets. **Bold** indicates the singular model that performed the best on the given task, whereas *italics* denotes the best models that were within the margin of error of each other for the given task. The overall column was computed by taking an average of individual task accuracies, weighted by the number of test examples per task.

in Table 2 with their corresponding callable unit and binding arguments.

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

5.2 Results

Language grounding accuracies for our two DRAGGN models, as well as the baseline Single-RNN, are presented in Table 3. All three models received the same set of training data, consisting of 2660 low-level action-oriented segments and 693 high-level goal-based sentences. All together, there are 17 unique combinations action-oriented callable units and respective binding arguments, and 6 unique combinations of goal-oriented callable units and binding arguments present in the data. Then, we evaluated all three models on the same set of held-out data, which consisted of 295 low-level segments and 86 high-level sentences.

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

5.3 Action Prediction

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

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

5.4 Goal Prediction

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

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

5.5 Unseen Action Prediction

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

6 Discussion

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

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

On action-oriented language, J-DRAGGN has grounding accuracy of around 20.2% while I-DRAGGN achieves a near-perfect 97.0%. Since J-DRAGGN only encodes the input language instruction once, the resulting vector representation is forced to characterize both callable unit and binding argument features. While this can result in positive information transfer and improve grounding accuracy in some cases (*e.g.* goal-based language), this enforced correlation heavily biases the model towards predicting combinations it has seen before. By learning separate representations for callable units and binding arguments, I-DRAGGN is able to generalize significantly better. This suggests that I-DRAGGN would perform best in situations where the instructional language consists of many disjoint words and phrases.

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

7 Conclusion

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

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

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

8 Acknowledgements

This material is based upon work supported by the National Science Foundation under grant number IIS-1637614 and the National Aeronautics and Space Administration under grant number NNX16AR61G.

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

References

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

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping

instructions to actions. In *Annual Meeting of the Association for Computational Linguistics*.

Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex. 2017. Accurately and efficiently interpreting human-robot instructions of varying granularities. *CoRR* abs/1704.06616.

R. Bellman. 1957. A Markovian decision process. *Indiana University Mathematics Journal* 6:679–684.

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

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

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

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

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

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

Sachithra Hemachandra, Felix Duvallet, Thomas M. Howard, Nicholas Roy, Anthony Stentz, and Matthew R. Walter. 2015. Learning models for following natural language directions in unknown environments. In *IEEE International Conference on Robotics and Automation*.

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

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

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

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

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

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

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI Conference on Artificial Intelligence*.

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

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

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

Martin L. Puterman. 1994. Markov decision processes: Discrete stochastic dynamic programming.

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

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

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