Learning to Interpret Natural Language Instructions

Monica Babeş-Vroman[+], **James MacGlashan**[*], **Ruoyuan Gao**[+], **Kevin Winner**[*]
Richard Adjogah[*], **Marie desJardins**[*], **Michael Littman**[+] and **Smaranda Muresan**[++]

[*] Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
[+] Computer Science Department, Rutgers University
[++] School of Communication and Information, Rutgers University

Abstract

This paper addresses the problem of training an artificial agent to follow verbal instructions representing high-level tasks using a set of instructions paired with demonstration traces of appropriate behavior. From this data, a mapping from instructions to tasks is learned, enabling the agent to carry out new instructions in novel environments.

1 Introduction

Learning to interpret language from a situated context has become a topic of much interest in recent years (Branavan et al., 2009; Branavan et al., 2010; Branavan et al., 2011; Clarke et al., 2010; Chen and Mooney, 2011; Vogel and Jurafsky, 2010; Goldwasser and Roth, 2011; Liang et al., 2011; Atrzi and Zettlemoyer, 2011; Tellex et al., 2011). Instead of using annotated training data consisting of sentences and their corresponding logical forms (Zettlemoyer and Collins, 2005; Kate and Mooney, 2006; Wong and Mooney, 2007; Zettlemoyer and Collins, 2009; Lu et al., 2008), most of these approaches leverage non-linguistic information from a situated context as their primary source of supervision. These approaches have been applied to various tasks such as following navigational instructions (Vogel and Jurafsky, 2010; Chen and Mooney, 2011; Tellex et al., 2011), software control (Branavan et al., 2009; Branavan et al., 2010), semantic parsing (Clarke et al., 2010; Liang et al., 2011) and learning to play games based on text (Branavan et al., 2011; Goldwasser and Roth, 2011).

In this paper, we present an approach to interpreting language instructions that describe *complex multipart tasks* by learning from pairs of instructions and behavioral traces containing a sequence of primitive actions that result in these instructions being properly followed. We do not assume a one-to-one mapping between instructions and primitive actions. Our approach uses three main subcomponents: (1) recognizing intentions from observed behavior using variations of Inverse Reinforcement Learning (IRL) methods; (2) translating instructions to task specifications using Semantic Parsing (SP) techniques; and (3) creating generalized task specifications to match user intentions using probabilistic Task Abstraction (TA) methods. We describe our system architecture and a learning scenario. We present preliminary results for a simplified version of our system that uses a unigram language model, minimal abstraction, and simple inverse reinforcement learning.

Early work on grounded language learning used features based on n-grams to represent the natural language input (Branavan et al., 2009; Vogel and Jurafsky, 2010). More recent methods have relied on a richer representation of linguistic data, such as syntactic dependency trees (Branavan et al., 2011; Goldwasser and Roth, 2011) and semantic templates (Tellex et al., 2011) to address the complexity of the natural language input. Our approach uses a flexible framework that allows us to incorporate various degrees of linguistic knowledge available at different stages in the learning process (e.g., from dependency relations to a full-fledged semantic model of the domain learned during training).

2 System Architecture

We represent tasks using the Object-oriented Markov Decision Process (OO-MDP) formalism (Diuk et al., 2008), an extension of Markov Decision Processes (MDPs) to explicitly capture relationships between objects. Specifically, OO-MDPs add a set of classes $\mathcal{C}$, each with a set of attributes $\mathcal{T}_{\mathcal{C}}$. Each OO-MDP state is defined by an unordered set of instantiated objects. In addition to these object definitions, an OO-MDP also defines a set of propositional functions that operate on objects. For instance, we might have a propositional function `toyIn(toy, room)` that operates on an object belonging to class "toy" and an object belonging to class "room," returning true if the specified "toy" object is in the specific "room" object. We extend OO-MDPs to include a set of *propositional function classes* ($\mathcal{F}$) associating propositional functions that describe similar properties. In the context of defining a task corresponding to a particular goal, an OO-MDP defines a subset of states $\beta \subset \mathcal{S}$ called *termination states* that end an action sequence and that need to be favored by the task's reward function.

Example Domain. To illustrate our approach, we present a simple domain called *Cleanup World*, a 2D grid world defined by various rooms that are connected by open doorways and can contain various objects (toys) that the agent can push around to different positions in the world. The Cleanup World domain can be represented as an OO-MDP with four object classes: agent, room, doorway, and toy, and a set of propositional functions that specify whether a toy is a specific shape (such as `isStar(toy)`), the color of a room (such as `isGreen(room)`), whether a toy is in a specific room (`toyIn(toy, room)`), and whether an agent is in a specific room (`agentIn(room)`). These functions belong to *shape*, *color*, *toy position* or *agent position* classes.

2.1 Interaction among IRL, SP and TA

The training data for the overall system is a set of pairs of verbal instructions and behavior. For example, one of these pairs could be the instruction *Push the star to the green room* with a demonstration of the task being accomplished in a specific environment containing various toys and rooms of different colors. We assume the availability of a set of features for each state represented using the OO-MDP propositional functions descibed previously. These features play an important role in defining the tasks to be learned. For example, a robot being taught to move furniture around would have information about whether or not it is currently carrying a piece of furniture, what piece of furniture it needs to be moving, which room it is currently in, which room contains each piece of furniture, etc. We present briefly the three components of our system (IRL, SP and TA) and how they interact with each other during learning.

Inverse Reinforcement Learning. Inverse Reinforcement Learning (Abbeel and Ng, 2004) addresses the task of learning a reward function from demonstrations of expert behavior and information about the state-transition function. Recently, more data-efficient IRL methods have been proposed, including the Maximum Likelihood Inverse Reinforcement Learning (Babeş-Vroman et al., 2011) or MLIRL approach, which our system builds on. Given even a small number of trajectories, MLIRL finds a weighting of the state features that (locally) maximizes the probability of these trajectories. In our system, these state features consist of one of the sets of propositional functions provided by the TA component. For a given task and a set of sets of state features, MLIRL evaluates the feature sets and returns to the TA component its assessment of the probabilities of the various sets.

Semantic Parsing. To address the problem of mapping instructions to semantic parses, we use a constraint-based grammar formalism, Lexicalized Well-Founded Grammar (LWFG), which has been shown to balance expressiveness with practical learnability results (Muresan and Rambow, 2007; Muresan, 2011). In LWFG, each string is associated with a syntactic-semantic representation, and the grammar rules have two types of constraints: one for semantic composition (Φ_c) and one for semantic interpretation (Φ_i). The semantic interpretation constraints, Φ_i, provide access to a semantic model (domain knowledge) during parsing. In the absence of a semantic model, however, the LWFG learnability result still holds. This fact is important if our agent is assumed to start with no knowledge of the task and domain. LWFG uses an ontology-based semantic representation, which is a logical form represented as a conjunction of atomic predicates. For example, the representation of the phrase *green room* is $\langle X_1.\textit{is}{=}\textit{green}, X.P_1 = X_1, X.\textit{isa}{=}\textit{room}\rangle$. The semantic representation specifies two concepts—`green` and `room`—connected through a property that can be uninstantiated in the absence of a semantic model, or instantiated via the Φ_i constraints to the property name (e.g, `color`) if such a model is present.

During the learning phase, the SP component, using an LWFG grammar that is learned offline, provides to TA the logical forms (i.e., the semantic parses, or the unlabeled dependency parses if no semantic model is given) for each verbal instruction. For example, for the instruction *Move the chair into the green room*, the semantic parser knows initially that *move* is a verb, *chair* and *room* are nouns, and *green* is an adjective. It also has grammar rules of the form S $\rightarrow$ Verb NP PP: Φ_{c1}, Φ_{i1},[1] but it has no knowledge of what these words mean (that is, to which concepts they map in the domain model). For this instruction, the LWFG parser returns the logical form:

$$\langle (X_1.\textit{isa}{=}\textit{move}, X_1.\textit{Arg1}{=}\ X_2)_{\textit{move}}, (X_2.\textit{det}{=}\textit{the})_{\textit{the}}, (X_2.\textit{isa}{=}\textit{chair})_{\textit{chair}}, (X_1.P_1 = X_3, P_2.\textit{isa}{=}\textit{into})_{\textit{into}}, (X_3.\textit{det}{=}\textit{the})_{\textit{the}}, (X_4.\textit{isa}{=}\textit{green}, X_3.P_2 = X_2)_{\textit{green}}, (X_3.\textit{isa}{=}\textit{room})_{\textit{room}}\rangle.$$

The subscripts for each atomic predicate indicate the word to which that predicate corresponds. This logical form corresponds to the simplified logical form `move(chair1,room1)`, `P1(room1,green)`, where predicate P1 is uninstantiated. A key advantage of this framework is that the LWFG parser has access to the domain (semantic) model via Φ_i constraints. As a result, when TA provides feedback about domain-specific meanings (i.e., groundings), the parser can incorporate those mappings via the Φ_i constraints (e.g., *move* might map to the predicate `blockToRoom` with a certain probability).

Task Abstraction. The termination conditions for an OO-MDP task can be defined in terms of the propositional functions. For example, the Cleanup World domain might include a task that requires the agent to put a specific toy (t_1) in a specific room (r_1). In this case, the termination states would be defined by states that satisfy $\texttt{toyIn}(t_1, r_1)$ and the reward function would be defined as $R_a(s, s') = \{1 : \texttt{toyIn}(t_1^{s'}, r_1^{s'}); -1 : \texttt{otherwise}\}$. However, such a task definition is overly specific and cannot be evaluated in a new environment that contains different objects. To remove this limitation, we define abstract task descriptions using *parametric lifted* reward and termination functions. A parametric lifted reward function is a first-order logic expression in which the propositional functions defining the reward can be selected as parameters. This representation allows much more general tasks to be defined; these tasks can be evaluated in any environment that contains the necessary object classes. For instance, the reward function for an abstract task that encourages an agent to take a toy of a certain shape to a room of a certain color (resulting in a reward of 1) would be represented as $R_a(s, s') = \{1 : \exists_{t^{s'} \in toy} \exists_{r^{s'} \in room} \texttt{P1}(t) \wedge \texttt{P2}(r) \wedge \texttt{toyIn}(t, r); -1 : \texttt{otherwise}\}$, where `P1` is a propositional function that operates on toy objects and `P2` is a propositional function that operates on room objects. An analogous definition can be made for termination conditions. Given the logical forms provided by SP, TA finds candidate tasks that might match each logical form, along with a set of possible *groundings* of those tasks. A grounding of an abstract task is the set of propositional functions to be applied to the specific objects in a given training instance. TA then passes these grounded propositional functions as the features to use in IRL. (If there are no candidate tasks, then it will pass all grounded propositional functions of the OO-MDP to IRL.) When IRL returns a reward function for these possible groundings and their likelihoods of representing the true reward function, TA determines whether any abstract tasks it has defined might match. If not, TA will either create a new abstract task that is consistent with the received reward functions or it will modify one of its existing definitions if doing so does not require significant changes. With IRL indicating the intended goal of a trace and with the abstract task indicating relevant parameters, TA can then inform SP of the task/domain specific meanings for the logical forms.

[1]For readability, we show here just the context-free backbone, without the augmented nonterminals or constraints.

A Learning from Scratch Scenario. Our system is trained using a set of sentence–trajectory pairs $((S_1, T_1), ..., (S_N, T_N))$. Initially, the system does not know what any of the words mean and there are no pre-existing abstract tasks. Let's assume that S_1 is *Push the star into the green room.*This sentence is first processed by the SP component, yielding the following logical forms: L_1 is $push(star1, room1), amod(room1, green)$ and L_2 is $push(star1), amod(room1, green)$, $into(star1, room1)$. These logical forms and their likelihoods are passed to the TA component, and TA induces incomplete abstract tasks, which define only the number and kinds of objects that are relevant to the corresponding reward function. TA can send to IRL a set of features involving these objects, together with T_1, the demonstration attached to S_1. This set of features might include: `agentTouchToy`(t_1), `toyIn`(t_1, r_1), `toyIn`(t_1, r_2), `agentIn`(r_1). IRL sends back a weighting of the features, and TA can select the subset of features that have the highest weights (e.g, (1.91, `toyIn`(t_1, r_1)), (1.12, `agentTouchToy`(t_1)), (0.80, `agentIn`(r_1)). Using information from SP and IRL, TA can now create a new abstract task, perhaps called `blockToRoom`, adjust the probabilities of the logical forms based on the relevant features obtained from IRL, and send these probabilities back to SP, enabling it to adjust its semantic model.

The entire system proceeds iteratively. While it is designed, not all features are fully implemented to be able to report experimental results. In the next section, we present a simplified version of our system and show preliminary results.

3 A Simplified Model and Experiments

In this section, we present a simplified version of our system with a unigram language model, inverse reinforcement learning and minimal abstraction. We call this version Model 0. The input to Model 0 is a set of verbal instructions paired with demonstrations of appropriate behavior. It uses an EM-style algorithm (Dempster et al., 1977) to estimate the probability distribution of words conditioned on reward functions (the parameters). With this information, when the system receives a new command, it can behave in a way that maximizes its reward given the posterior probabilities of the possible reward functions given the words.

Algorithm 1 shows our EM-style Model 0. For all possible reward–demonstration pairs, the E-step of EM estimates $z_{ji} = \Pr(R_j|(S_i, T_i))$, the probability that reward function R_j produced sentence-trajectory pair (S_i, T_i), This estimate is given by the equation below:

$$zji = \Pr(R_j|(S_i, T_i)) = \frac{\Pr(R_j)}{\Pr(S_i, T_i)} \Pr((S_i, T_i)|R_j)$$

$$= \frac{\Pr(R_j)}{\Pr(S_i, T_i)} \Pr(T_i|R_j) \prod_{w_k \in S_i} \Pr(w_k|R_j)$$

where S_i is the i^{th} sentence, T_i is the trajectory demonstrated for verbal command S_i, and w_k is an element in the set of all possible words (vocabulary). If the reward functions R_j are known ahead of time, $\Pr(T_i|R_j)$ can be obtained directly by solving the MDP and estimating the probability of trajectory T_i under a Boltzmann policy with respect to R_j. If the R_js are not known, EM can estimate them by running IRL during the M-step (Babeş-Vroman et al., 2011).

The M-step in Algorithm 1 uses the current estimates of z_{ji} to further refine the probabilities $x_{kj} = \Pr(w_k|R_j)$:

$$x_{kj} = \Pr(w_k|R_j) = \frac{1}{X} \frac{\Sigma_{w_k \in S_i} \Pr(R_j|S_i) + \epsilon}{\Sigma_i N(S_i) z_{ji} + \epsilon}$$

where ϵ is a smoothing parameter, X is a normalizing factor and $N(S_i)$ is the number of words in sentence S_i.

To illustrate our Model 0 performance, we selected as training data six sentences for two tasks (three sentences for each task) from a dataset we have collected using Amazon Mechanical Turk for the Cleanup Domain. We show the training data in Figure 1. We obtained the reward function for each task using MLIRL, computed the $\Pr(T_i|R_j)$, then ran Algorithm 1 and obtained the parameters $\Pr(w_k|R_j)$. After this training process, we presented the agent with a new task. She is given the instruction S_N: *Go to green room.* and a starting state, somewhere in the same grid. Using parameters $\Pr(w_k|R_j)$, the agent can estimate:

Algorithm 1 EM-style Model 0

```
  Input: Demonstrations $\{(S_1, T_1), ..., (S_N, T_N)\}$, number of reward functions $J$, size of vocabulary $K$.
  Initialize: $x_{11}, \ldots, x_{JK}$, randomly.
  repeat
    E Step: Compute
    $z_{ji} = \frac{\Pr(R_j)}{\Pr(S_i,T_i)} \Pr(T_i|R_j) \prod_{w_k \in S_i} x_{kj}$.

    M step: Compute
    $x_{kj} = \frac{1}{X} \frac{\Sigma_{w_k \in S_i} \Pr(R_j|S_i)+\epsilon}{\Sigma_i N(S_i) z_{ji}+\epsilon}$.
  until target number of iterations completed.
```

$\Pr(S_N|R_1) = \prod_{w_k \in S_N} \Pr(w_k|R_1) = 8.6 \times 10^{-7}$, $\Pr(S_N|R_2) = \prod_{w_k \in S_N} \Pr(w_k|R_2) = 4.1 \times 10^{-4}$, and choose the optimal policy corresponding to reward R_2, thus successfully carrying out the task. Note that R_1 and R_2 corresponded to the two target tasks, but this mapping was determined by EM. We illustrate the limitation of the unigram model by telling the trained agent to *Go with the star to green*, (we label this sentence S'_N). Using the learned parameters, the agent computes the following estimates:
$\Pr(S'_N|R_1) = \prod_{w_k \in S'_N} \Pr(w_k|R_1) = 8.25 \times 10^{-7}$, $\Pr(S'_N|R_2) = \prod_{w_k \in S'_N} \Pr(w_k|R_2) = 2.10 \times 10^{-5}$. The agent wrongly chooses reward R_2 and goes to the green room instead of taking the star to the green room. The problem with the unigram model in this case is that it gives too much weight to word frequencies (in this case *go*) without taking into account what the words mean or how they are used in the context of the sentence. Using the system described in Section 2, we can address these problems and also move towards more complex scenarios.

4 Conclusions and Future Work

We have presented a three-component architecture for interpreting natural language instructions, where the learner has access to natural language input and demonstrations of appropriate behavior. Our future work includes fully implementing the system to be able to build abstract tasks from language information and feature relevance.

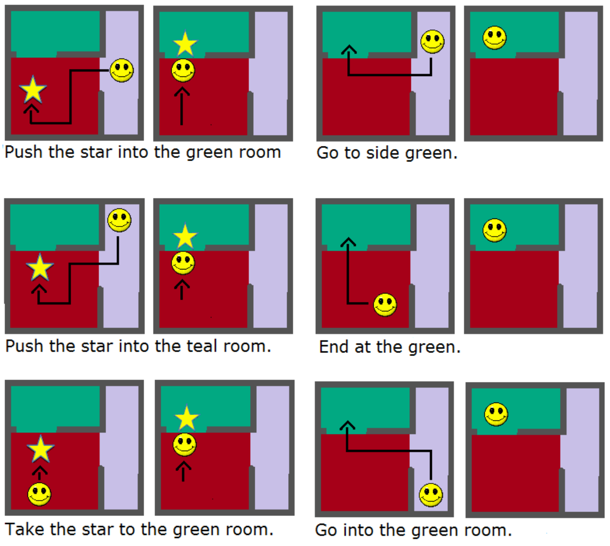
Push the star into the green room
Go to side green.
Push the star into the teal room.
End at the green.
Take the star to the green room.
Go into the green room.

Figure 1: Training data for 2 tasks: Taking the star to the green room (left) and Going to the green room (right).

Acknowledgments

The authors acknowledge the support of the National Science Foundation (collaborative grant IIS-00006577 and IIS-1065195). The authors thank the anonymous reviewers for their feedback. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organization.

References

Pieter Abbeel and Andrew Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference in Machine Learning (ICML 2004)*.

Yoav Atrzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers for conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*.

Monica Babeş-Vroman, Vukosi Marivate, Kaushik Subramanian, and Michael Littman. 2011. Apprenticeship learning about multiple intentions. In *Proceedings of the Twenty Eighth International Conference on Machine Learning (ICML 2011)*.

S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on*

Natural Language Processing of the AFNLP: Volume 1 - Volume 1, ACL '09.

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

S.R.K. Branavan, David Silver, and Regina Barzilay. 2011. Learning to win by reading manuals in a monte-carlo framework. In *Association for Computational Linguistics (ACL 2011)*.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011).*, pages 859–865.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Association for Computational Linguistics (ACL 2010)*.

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

Carlos Diuk, Andre Cohen, and Michael Littman. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*.

Dan Goldwasser and Dan Roth. 2011. Learning from natural instructions. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL 2011)*.

Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08.

Smaranda Muresan and Owen Rambow. 2007. Grammar approximation by representative sublanguage: A new model for language learning. In *Proceedings of ACL*.

Smaranda Muresan. 2011. Learning for deep language understanding. In *Proceedings of IJCAI-11*.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Articifical Intelligence.*

Adam Vogel and Dan Jurafsky. 2010. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL 2010)*.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI-05*.

Luke Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Association for Computational Linguistics (ACL'09)*.