Learning to Interpret Natural Language Instructions

Abstract

We address the problem of training an artificial agent to follow verbal instructions 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

In this paper, we address the following problem: given a verbal instruction, what is the sequence of actions that the instructed agent needs to perform to successfully carry out the corresponding task? The goal of our project is to develop techniques for a computer or robot to learn from examples to carry out multipart tasks, specified in natural language, on behalf of a user. 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 Natural Language Processing (NLP) techniques; and (3) creating generalized task specifications to match user intentions, using probabilistic Task Abstraction (TA) methods.

In the next section, we discuss related work in this general problem domain. We then describe the architecture of our system and show results from a preliminary experiment. Finally, we summarize our ongoing and future research on this problem.

2 Related Work

Our work relates to the broad class of methods that aim to learn to interpret language from a situated context (Branavan et al., 2009; Branavan et al., 2010; Branavan et al., 2011; Clarke et al., 2010; Chen and Mooney, 2011; Vogel and Jurafsky, 2010; Grubb et al., 2011; Goldwasser and Roth, 2011; Liang et al., 2011; Hewlett et al., 2010; Tellex et al., 2011; Atrzi and Zettlemoyer, 2011). Instead of using annotated training data consisting of sentences and their corresponding logical forms (Kate and Mooney, 2006; Wong and Mooney, 2007; Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2009), 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: interpreting verbal commands in the context of navigational instructions (Vogel and Jurafsky, 2010; Chen and Mooney, 2011; Grubb et al., 2011), robot manipulation (Tellex et al., 2011), puzzle solving and software control (Branavan et al., 2009; Branavan et al., 2010); semantic parsing (Clarke et al., 2010; Liang et al., 2011; Atrzi and Zettlemoyer, 2011), learning game strategies from text (Branavan et al., 2011), and inducing knowledge about a domain based on text (Goldwasser and Roth, 2011). The task closest to ours is interpreting navigation instructions. However, our goal is to move away from low-level instructions that correspond directly to actions in the environment (Branavan et al., 2009; Vogel and Jurafsky, 2010) to high-level task descriptions expressed using complex language.

Early work on grounded language learning used the bag-of-words approach to represent the natural language input (Branavan et al., 2009; Branavan et al., 2010; 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 (Grubb et al., 2011; 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 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).

3 Background and 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. In the context of defining a task corresponding to a particular goal, an OO-
MDP defines a subset of states $\beta \subseteq S$, called termination states that end an action sequence and that need to be favored by the task’s reward function. We extend OO-MDPs to include a set of propositional function classes ($F$) associating propositional functions that describe similar properties.

To illustrate our approach, we present a simple Sokoban-style 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 agent can move north, south, east, or west, unless a wall prevents her from doing so. To move a toy, she must be in an adjacent cell and move in the direction of the toy, resulting in the toy being pushed to the opposite adjacent cell from the agent (unless a wall or another toy is in the way, in which case nothing happens).

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 $\text{isStar}(\text{toy})$), the color of a room (such as $\text{isGreen}(\text{room})$), whether a toy is in a specific room ($\text{toyIn}(\text{toy, room})$), and whether an agent is in a specific room ($\text{agentIn}(\text{room})$). These functions belong to respective shape, color, and toy or agent position classes.

We assume the availability of a set of features for each state represented using these propositional functions. 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. Next, we show the role of each of the system components and how they interact with each other.

### 3.1 Natural Language Processing

The Natural Language Processing component addresses the problem of mapping instructions to semantic parses, building on recent work on learning to map natural language utterances to meaning representations. The core of this approach is a recently developed grammar formalism, Lexicalized Well-Founded Grammar (LWFG), which balances expressiveness with practical—and provable—learnability results (Muresan and Ramshaw, 2007; Muresan, 2010; Muresan, 2011). 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 $\langle X_1, \text{is}=\text{green, } X.P_1 = X_1, X.\text{isa}=\text{room} \rangle$. 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 constraints to the property name (e.g. color) if such a model is present.

During the learning phase, the NLP component gives the logical forms for each verbal instruction to TA, using a LWFG grammar that is learned offline (i.e., the semantic parses, or the unlabeled dependency parses if no semantic model is given). For example, for the instruction Move the chair into the green room, the NLP component 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 $S \rightarrow \text{Verb NP PP}$ but it has no knowledge of what these words mean (that is, to which concepts they map in the domain model). The logical form produced by LWFG corresponds to the simplified logical form $\text{move(\text{chair1, room1}, P_1(\text{room1, green})}$, where predicate $P_1$ is uninstantiated. A key advantage of this framework is that the LWFG parser has access to the domain (semantic) model via semantic interpretation constraints, $\Phi_i$. As a result, when TA provides feedback about domain-specific meanings (i.e., groundings), the parser can incorporate those mappings via the constraints (e.g., $\text{move}$ might map to the predicate “MoveToRoom” with a certain probability).

### 3.2 Task Abstraction

Given the logical forms provided by NLP, 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 (parameters of the abstract task) 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 NLP of the semantics of its logical parse. The entire system proceeds iteratively, with each component, directly or indirectly, informing the others.

3.3 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 (Babes-Vroman et al., 2011) or MLIRL approach, which our system builds on. Given a small number of trajectories, MLIRL finds a weighing 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 TA. 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.

3.4 A Simplified System Example

In this section, we show a simplified version of our system with a unigram language model and minimal abstraction. We call this version Model 0. The input to Model 0 is as described: a set of verbal instructions paired with demonstrations of appropriate behavior. It uses the Expectation Maximization 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 EM Model 0

Input: Demonstrations \{(S_1,T_1), \ldots, (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
    \[ x_{kj} = \frac{1}{X} \frac{\sum_{w_k \in S_i} Pr(R_j|S_i) + \epsilon}{\sum_{S_i} N(S_i) z_{ji} + \epsilon}. \]
until target number of iterations completed.

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:

\[
z_{ji} = Pr(R_j|(S_i,T_i)) = \frac{Pr(R_j)}{Pr(S_i,T_i)} Pr((S_i,T_i)|R_j)
= \frac{Pr(R_j)}{Pr(S_i,T_i)} Pr(T_i|R_j) \prod_{w_k \in S_i} Pr(w_k|R_j)
\]

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 \(R_j\)'s are not known, EM can estimate them by running IRL during the M-step (Babes-Vroman et al., 2011).

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

\[
x_{kj} = Pr(w_k|R_j) = \frac{1}{X} \frac{\sum_{w_k \in S_i} Pr(R_j|S_i) + \epsilon}{\sum_{S_i} N(S_i) z_{ji} + \epsilon}
\]

where \(\epsilon\) is a smoothing parameter, \(X\) is a normalizing factor and \(N(S_i)\) is the number of words in sentence \(S_i\). We show these steps in Algorithm 1.

4 Preliminary Results

While our system is fully designed, it is still being implemented. As such, we only have results for spe-
cific parts of the system. We illustrate a simplified version of our system in the Cleanup Domain below. To collect a corpus of training data that would be linguistically interesting, we crowdsourced the task of generating instructions for example trajectories using Amazon Turk. Example trajectories were presented to users as an animated image of the agent interacting in the world, and users were asked to provide a corresponding instruction. This process had predictably mixed results: about 1/3 of the resulting instructions were badly malformed or inappropriate. For the results shown here, we have used “human-inspired” sentences, consisting of a manually constructed subset of sentences we received from our Turk experiment. These sentences were additionally simplified and clarified by retaining only the last verb and by pruning irrelevant portions of the sentence. Instructions are typically of the form, “Move the green star to the red room”; the trajectories in the training data consist of a sequence of states and actions that could be performed by the agent to achieve this goal.

To illustrate the EM unigram model, we selected six random sentences for two tasks (three sentences for each task). 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 \( S_N^\prime \): “Go to green room.” and a starting state, somewhere in the same grid. Using parameters \( \Pr(w_k | R_j) \), the agent can estimate:

\[
\Pr(S_N^\prime | R_1) = \prod_{w_k \in S_N^\prime} \Pr(w_k | R_1) = 8.6 \times 10^{-7},
\]

\[
\Pr(S_N^\prime | R_2) = \prod_{w_k \in S_N^\prime} \Pr(w_k | R_2) = 4.1 \times 10^{-4}
\]

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.

Using a minimalistic abstraction model, the Model 0 agent could learn that words like “green” and “teal” map to the same abstract color. Still, this agent is very limited and could benefit from a richer abstraction and a language model that can capture semantic information. Our aim is to have an agent learn what words mean and how they map to objects and their attributes in her environment.

We illustrate the limitation of the unigram model by telling the trained agent to “Go with the star to green.” (we label this sentence \( S_N^\prime \)). Using the learned parameters, the agent will compute the following estimates:

\[
\Pr(S_N^\prime | R_1) = \prod_{w_k \in S_N^\prime} \Pr(w_k | R_1) = 8.25 \times 10^{-7},
\]

\[
\Pr(S_N^\prime | R_2) = \prod_{w_k \in S_N^\prime} \Pr(w_k | R_2) = 2.10 \times 10^{-5}
\]

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. We will try to address some of these issues in the future by bringing to bear more complex language and abstraction models, as described in Section 3.

5 Conclusions and Future Work

Our project grounds language in a simulated environment by training an agent from verbal commands paired with demonstration of appropriate behavior, using three major components. Our future work includes fully implementing the system, to make it able to build abstract tasks from language information and feature relevance. We also believe that learning semantics will enable our system to carry out commands that the system has not seen before.
References


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).


