Instructable Intelligent Personal Agent

Amos Azaria[1], **Jayant Krishnamurthy**[2], **Tom M. Mitchell**[1]
[1] Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA 15213
[2] Allen Institute for Artificial Intelligence, Seattle, WA 98103
{azariaa, jayantk, tom.mitchell}@cs.cmu.edu

Abstract

Unlike traditional machine learning methods, humans often learn from natural language instruction. As users become increasingly accustomed to interacting with mobile devices using speech, their interest in instructing these devices in natural language is likely to grow.

We introduce our Learning by Instruction Agent (LIA), an intelligent personal agent that users can teach to perform new action sequences to achieve new commands, using solely natural language interaction. LIA uses a CCG semantic parser to ground the semantics of each command in terms of primitive executable procedures defining sensors and effectors of the agent. Given a natural language command that LIA does not understand, it prompts the user to explain how to achieve the command through a sequence of steps, also specified in natural language. A novel lexicon induction algorithm enables LIA to generalize across taught commands, e.g., having been taught how to "forward an email to Alice," LIA can correctly interpret the command "forward this email to Bob." A user study involving email tasks demonstrates that users voluntarily teach LIA new commands, and that these taught commands significantly reduce task completion time. These results demonstrate the potential of natural language instruction as a significant, under-explored paradigm for machine learning.

Introduction

Natural language instructions are commonly used to teach humans but are rarely used to teach machine learning systems. For example, a user may tell his human assistant "I'm stuck in traffic and will be late." The assistant may not know what to do, so the user may then explain "first, use GPS to estimate my time of arrival, then see who I am meeting and send them an email indicating that I'll be late." Having been instructed thus, the assistant now understands how to handle similar situations in the future. We would like users to be able to teach machine learning systems through this kind of natural language interaction.

As a starting point, we focus on building an instructable intelligent personal agent. Intelligent personal agents – such as Siri, Google Now, Cortana and Echo – have demonstrated great potential for assisting users with basic daily tasks. However, unlike the instructable human assistant above, interactions with these agents are limited to commands preprogrammed by the developers. A teachable agent has two significant benefits over such preprogrammed agents. First, teaching enables a user to define a new command and a sequence of actions to perform it, including commands that were not considered by the software developer. Second, teaching can also be used to expand the range of natural language phrasings understood by the system (e.g., by teaching the system that "drop a note to Bill" has the same meaning as "send an email to Bill".). Together, these two capabilities might enable a community of users to jointly instruct a system to achieve a significant number of new commands, and ways to express them in natural language.

This paper presents a Learning by Instruction Agent (LIA), an intelligent agent that allows users to teach it new commands using solely natural language interactions. LIA operates in an email environment, where it learns to interpret natural language commands in terms of given primitive actions such as sending emails. Users teach LIA to perform a new command by providing step-by-step instructions. LIA is then able to generalize and later execute this command with different parameters; for example, having been taught "forward an email to Alice," LIA can correctly interpret "forward this email to Bob." Users can also teach LIA declarative knowledge by defining new concepts and relations between them, e.g., defining the notion of a contact, and that each contact has an associated email address. The contributions of this work are:

1. A working implementation of an agent that can be taught in natural language to perform new commands.
2. A novel lexicon induction algorithm that enables the agent to generalize from a taught command to unseen commands.
3. A user study demonstrating that users voluntarily teach the agent new commands, and that these taught commands result in a significant time savings on future tasks.

Related work

Three fields with vast work are closely related to ours showing that researchers are reaching out to the vision of an instructable agent. The first is grounded language acquisition (Harnad 1990; Chen and Mooney 2008). In this

---

field, the main challenges are extracting the semantic meaning of words and sentences and connecting language to action and perception. One example is work by Chen and Mooney (2008), in which a system learns how to sportscast a RoboCup simulated game and outputs statements, for example, that a certain player has a ball, that the ball was passed on to a different player and that a player has intercepted this pass. The system learns from data which includes many games and the text description of what is shown in the game, not having any additional domain or language background. The system builds a generative model based on PCFG, which maximizes the likelihood of the data, and uses this model to sportcast a new, unseen, game. Chen and Mooney, also apply a similar method to the domain of navigation (Chen and Mooney 2011). In this domain, a user must navigate via a 3-D maze with objects located in it and reach a specific goal. The data in this domain is composed of a set of paragraphs in natural language, each associated with a video which shows the execution of this paragraph. This learning method may be categorized as learning by observation. In recent work Thomason et al. (2015) use CCG parsing on natural language commands in order to execute them by a robot. They use conversations with previous users to better understand user commands as well as overcoming typos and spelling mistakes.

A second related research area is programming by demonstration (PbD) also referred to as learning from demonstration (LfD) and imitation learning. Much of this research derives from the human-robot interaction (HRI) community (although some work does not involve robots (Myers et al. 1991; Allen et al. 2007)). The common case study in PbD is with a person trying to show a robot how to lift or select a certain object, move an object or perform some other task (Argall et al. 2009). In most studies the human teacher actually moves the robot's arms to perform the taught task (Calinon, Guenter, and Billard 2007), or controls the robot using a control peg, while in some other studies, the human teacher performs the task in front of the robot's cameras (Nakaoka et al. 2007), or wears data gloves (Kuklinski et al. 2014). In many cases the robot can also generalize beyond the specific training scenario, to perform the task also in different conditions. For example, Calinon et al. (2007), teach a robot by demonstration how to move a chess piece on a board of chess by moving the robot's arms. The robot can then generalize and move the same piece also when it is located in a different location. Billard et al. (2008) state that learning by demonstration has three advantages, the first is that demonstration reduces the search space for the robot, allowing the robot to find the way to perform the task. The third, is that PbD couples the perception (or vision) and action (or robot motors). While, programming by demonstration usually requires several training examples, Allen et al. (2007) build a system with a complex graphical interface which poses questions to the teacher as it learns a new task (in the information retrieval domain). Their system succeeded in learning 30 out of 55 tasks based on a single demonstration by a human subject for each task. However, as they state, subjects did not find the system easy to use, and each subject required a full work day to teach approximately 3 tasks. Programming by demonstration was also used in authoring intelligent tutor behavior. Koedinger et al. (2004) built a system with a graphical interface which allowed an author to create arithmetic problems and demonstrate their solutions as well as common errors. The intelligent tutor created a solution graph and allowed the author to annotate this graph with hints and feedback. The intelligent tutor used these examples to tutor a student providing hints and feedback.

The third field which is related to this research is natural language programming (Biermann 1983), in which a programmer can use natural language to develop software. In Inform 7 (Reed 2010), for example, a programmer can create an interactive fiction program using statements which are actually valid English sentences. Some examples for such statements in the Inform 7 language are "The kitchen is a room," "The kitchen has a stove," and "The description of the stove is: 'very dirty' ". However, although being in natural language, statements in Inform 7, are required to be in a specific form. Therefore, while one may not be required to know the specific grammar of the natural language program language to read a program written in that language, it is required to have programming skills and familiarity with the given program language in order to actually develop a program in that language. There has also been some preliminary work on translating functionality descriptions in natural language into scripts by Le et al. (2013), and translation of conditions into if-then clauses (Quirk, Mooney, and Galley 2015).

The Agent (LIA)

LIA, our instructable agent operates in an email domain, where the basic actions include reading, composing, and sending emails. We chose this domain as our research environment because email manipulation is a common use of mobile phones where we anticipate that an instructable agent may be useful. The user interacts with LIA in a text dialogue by giving it commands in natural language, and the agent responds both in natural language and by taking various actions. LIA is built from two components that enable it to intelligently respond to user commands: a semantic parser which assigns executable semantics to each natural language command, and a back-end which executes these commands. The back-end contains a number of built-in, executable functions understood by LIA, such as `sendEmail`, along with a declarative knowledge base containing statements in predicate logic. LIA interprets commands using a semantic parser that maps each command to a *logical form* – a program in a Lisp-like language – containing one or more of these functions and predicates. This logical form represents the semantics of the user command, and is evaluated (executed) by the back-end to produce a response.

LIA can be instructed in two distinct fashions, corresponding to two distinct types of knowledge that LIA can acquire. First, the user can teach LIA new /em declarative knowledge by defining new concepts, along with fields and instances of those concepts. For example, the user can define the concept "contact" and further state that "a contact has an email address" and "bob is a contact." LIA processes

Table 1: Training examples for the semantic parser consist of a natural language command paired with a logical form, which is a computer program written in a Lisp-like language composed of one or more basic actions that LIA can perform. The top four examples are part of the default training set and the bottommost example was generated by a user teaching interaction.

Text Command	Logical Form
set the subject to time to go	(setFieldFromString (getMutableFieldByFieldName subject) (stringValue “time to go”))
send the email	(send email)
set body to email’s body and send email	(doSeq (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body))) (send email))
add length as a field in table	(addFieldToConcept table (stringNoun “length”))
forward to charlie	(doSeq (doSeq (doSeq (doSeq (createInstanceByConceptName outgoingemail) (setFieldFromFieldVal (getMutableFieldByFieldName subject) (evalField (getFieldByInstanceNameAndFieldName email subject)))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient) (evalField (getFieldByInstanceNameAndFieldName charlie email)))) (sendEmail))

Table 2: Lexicon entries required to parse the examples in Table 1. The syntactic category specifies how each word can combine with adjacent words and phrases during parsing and the logical form gives the word’s meaning. The bottommost entry was automatically produced by lexicon induction when LIA was taught how to “forward to charlie.”

Word	Syntactic Category	Logical Form
set	((S/PP_StringV)/MutableField)	(lambda x y (setFieldFromString x y))
to	PP_StringV/StringV	(lambda x x)
subject	FieldName	subject
send	S/InstanceName	(lambda x (send x))
email	InstanceName	email
set	((S/PP_FieldVal)/MutableField)	(lambda x y (setFieldFromFieldVal x y))
to	PP_FieldVal/FieldVal	(lambda x x)
and	(S/S)\S	(lambda x y (doSeq x y))
’s	((Field\InstanceName)/FieldName)	(lambda x y (getFieldByInstanceNameAndFieldName y x))
forward	(S/InstanceName)	(lambda x (doSeq (doSeq (doSeq (doSeq (createInstanceByConceptName outgoingemail) (setFieldFromFieldVal (getMutableFieldByFieldName subject) (evalField (getFieldByInstanceNameAndFieldName email subject)))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient) (evalField (getFieldByInstanceNameAndFieldName x email)))) (sendEmail)))

these interactions by adding new concepts, fields and instances to its knowledge base (as performed in (Haas and Hendrix 1980)). Second, using our novel approach, the user can teach LIA new /em procedural knowledge, i.e., how to execute a new command. For example, the user can teach the system how to “forward” an email by providing natural language instructions that map to actions the system already understands. LIA learns new procedural knowledge from these interactions using a novel lexicon induction algorithm that updates the semantic parser. The updated parser is then able to understand both the taught command as well as unseen, but similar commands.

Back-end Command Executor

LIA has a back-end that can evaluate logical forms (lambda expressions) incorporating 45 primitive, executable functions. Some examples are sendEmail, which sends the composed email, setFieldFromFieldVal, which sets a field from an evaluation of a different field, addFieldToConcept which adds a field to a concept, createInstanceByConceptName which creates an instance and deleteInstance which deletes an instance. Two additional notable functions that will be discussed later are unknownCommand and teachNewCommand, both of which start a dialog that lets the user teach a new command.

Upon execution, the back-end also builds a user-friendly response that either indicates which action was just per-

Syntactic Input	Syntactic Output	Semantics
FieldName	MutableField	(lambda x (getMutableFieldByFieldName x))
Field	FieldVal	(lambda x (evalField x))

Table 3: Examples of unary rules (required to parse the examples in Table 1).

formed (e.g., "The subject field of the outgoing email was set to party time for all"), or, in case of failure provides an informative statement conveying the reason for failure and sometimes suggesting a possible correction. For example, if the user says "set momthebest7@bestforyou.com," LIA replies "Sorry, but I don't know what should be set to momthebest7@bestforyou.com. Please repeat and tell me what should be set to it (e.g., set example to momthebest7@bestforyou.com)."

Semantic Parser

LIA uses a Combinatory Categorial Grammar (CCG) semantic parser to map natural language commands to logical forms containing functions and concepts executable by the back-end. CCG is often used to build semantic parsers due to its tight coupling of syntax and semantics (Zettlemoyer and Collins 2005). CCG grammars are more expressive than context-free grammars, and are able to represent long-range dependencies present in some linguistic constructions, such as relative clauses, that cannot be represented in context-free formalisms (Steedman and Baldridge 2011).

A CCG semantic parser has three parts: a lexicon, a set of grammar rules, and a trained parameter vector. The lexicon is a table mapping words to syntactic categories and logical forms (see Table 2). The intuition of CCG is that, syntactically and semantically, *words behave like functions*. Thus, syntactic categories represent function type specifications, where the argument type appears on the right of the slash and the return type on the left. The direction of the slash determines on which side of the syntactic category each argument must appear. For example, the syntactic category ((S\PP_StringV)/MutableField) accepts a MutableField on the right, followed by a PP_StringV on the left, and returns an S. In this fashion, the syntactic category of a lexicon entry specifies how it can combine with other words during parsing. Our CCG parser also permits strings from the command to enter the parse with the syntactic category StringN or StringV, and words in the sentence to be skipped. This second capability is used to ignore function words in the sentence that contribute little to the overall meaning.

Parsing in CCG derives syntactic categories and logical forms for phrases from their constituent parts by applying a small number of grammar rules. These rules correspond to standard function operations, such as application and composition. For example, the lexicon entries for "send" and "email" in Table 2 can be combined using function application to derive the second example in Table 1. Our grammar also includes a small number of unary rules that represent common implicit conversions between types.

Together, the lexicon and the grammar rules define a set of possible parses for every input command, each of which may have a different logical form. In order to select a single best parse, the semantic parser is trained using a data set of commands paired with their corresponding logical forms (see Table 1). First, we define a feature function ϕ that maps a CCG parse t of a command s to a feature vector $\phi(t, s)$. Our features include indicator features for the lexicon entries used in the parse, the parse's function/argument applications, and various features derived from the string itself. During training, the parser learns a parameter vector θ that assigns a high score $\theta^T \phi(t, s)$ to correct parses. At test time, the parser selects the highest-scoring parse for each command, i.e., the parse t that maximizes $\theta^T \phi(t, s)$. For more information about CCG semantic parsing, including details of parsing and training algorithms, we refer the reader to Zettlemoyer and Collins (2005).

LIA's semantic parser has over 300 lexicon entries, 14 unary rules, and was trained using 150 training examples, and Table 3 the required unary rules.

Logical Form Evaluation

The logical forms output by the semantic parser are evaluated by the back-end in standard Lisp fashion. Each argument of a function application is recursively evaluated in left-to-right order. For example, to interpret the first example command in Table 1, LIA first evaluates getMutableFieldByFieldName and subject, then performs a function application with the corresponding values. The result of this application is the subject of the outgoing email. Next, LIA evaluates the expression containing stringValue, which returns the string "time to go." Finally, LIA calls setFieldFromString with the outgoing email's subject and the string "time to go." The result of evaluation is to set the outgoing email's subject to "time to go."

Learning New Commands

LIA learns new commands through an instruction dialogue that is initiated whenever the user enters a command that LIA does not understand. In this case, the semantic parser outputs the logical form (unknownCommand). Evaluating this expression starts an instruction interaction that enables the user to teach LIA how to execute the given command. LIA asks the user what it should do first in order to execute the command being learned, and then asks for consecutive actions, one at a time. The user responds with a sequence of natural language commands that LIA should perform. LIA parses and evaluates each command in the sequence during instruction to confirm that these commands can be performed. Once the user ends the instruction interaction, LIA has a sequence of logical forms that, when evaluated sequentially, produce the desired result for the given command. LIA combines this sequence into a single logical form using the function doSeq (see Table 1, bottom).

This alone does not allow the agent to generalize beyond this training command. However, LIA uses a novel *lexicon induction* algorithm to update the semantic parser to generalize the instruction to interpret other, similar commands. This algorithm learns which words in the taught command correspond to each part of the complete logical form. It first parses the taught command with the current semantic parser and examines the 100 best parses of each span of the command. If a span's logical form is a subexpression of the complete logical form, then it could be an argument that should be filled during parsing. For example, in the command "forward to charlie," the text span "charlie" parses to the logical form charlie, which is a subexpression of the complete logical form (Table 1, bottom). The algorithm finds possible arguments and removes them from the complete logical form to construct a set of candidate logical forms, then creates lexicon entries by pairing each of these candidates with every non-stopword in the command. In our example, the algorithm creates the final entry in Table 2, where charlie has been extracted as an argument to "forward." These lexicon entries are added to the lexicon, the command/logical form pair is added to the training set, and the parser is re-trained. Pseudocode for lexicon induction is provided as Algorithm 1.

Algorithm 1 Lexicon induction for a taught command.

```
Input: $s$ - A command containing $n$ tokens.
    $\ell$ - The user-provided logical form for the command.
    $b$ - Beam size.
Output: $\Lambda$ - A set of induced lexicon entries.
    Phase 1: Find spans of $s$ that can be parsed to logical
    forms that are subexpressions of $\ell$.
Parse $s$ with beam search to produce $b$ logical forms
    $\lambda_{i,j,k}$ for every span $(i, j)$, $(k \in [1, ..., b])$ .
$C \leftarrow \{\}$ // Set of candidate spans
for each span $(i, j) : 0 \leq i < j \leq n$ do
    for $k : 1 < k \leq b$ do
        if ISSUBEXPRESSION$(\lambda_{i,j,k}, \ell)$ then
            $C \leftarrow C \cup \{(i, j, k)\}$
Discard elements of $C$ whose sentence span is com-
    pletely contained by a larger candidate.
    Phase 2: Generate lexicon entries by extracting argu-
    ments in $C$ from $\ell$.
$\Lambda \leftarrow \{\}$
for each subset $S$ of $C$ with non-overlapping spans do
    $\ell' \leftarrow \ell$
    for each $(i, j, k) \in S$ do
        // Replace $\lambda_{i,j,k}$ in $\ell'$ with an argument variable $a$.
        $a \leftarrow$ GETUNIQUEVARIABLENAME()
        body $\leftarrow$ SUBSTITUTE$(\lambda_{i,j,k}, \ell', a)$
        $\ell' \leftarrow$ (lambda $a$ body)
    for each $m : 0 \leq m < n$ do
        if $m$ is not contained by a span in $S$ and $s_m$ is not
        a stopword then
            $\Lambda \leftarrow \Lambda \cup \{s_m := \ell'\}$
```

The use of (unknownCommand) to begin the teaching interaction allows the user to provide all commands naturally, triggering a teaching interaction only when the user enters language the parser does not understand. It also ensures that taught commands are written the same way that the user wants to use them. LIA also allows the user to explicitly initiate a teaching interaction by using a command such as "teach a command," which is parsed to (teachNewCommand). In this case, LIA responds "I'm happy to hear that want to teach me a new command. Now say the command the way you would use it , then I will ask you what exactly to do in that case. I will try to generalize to similar sentences."

Evaluation

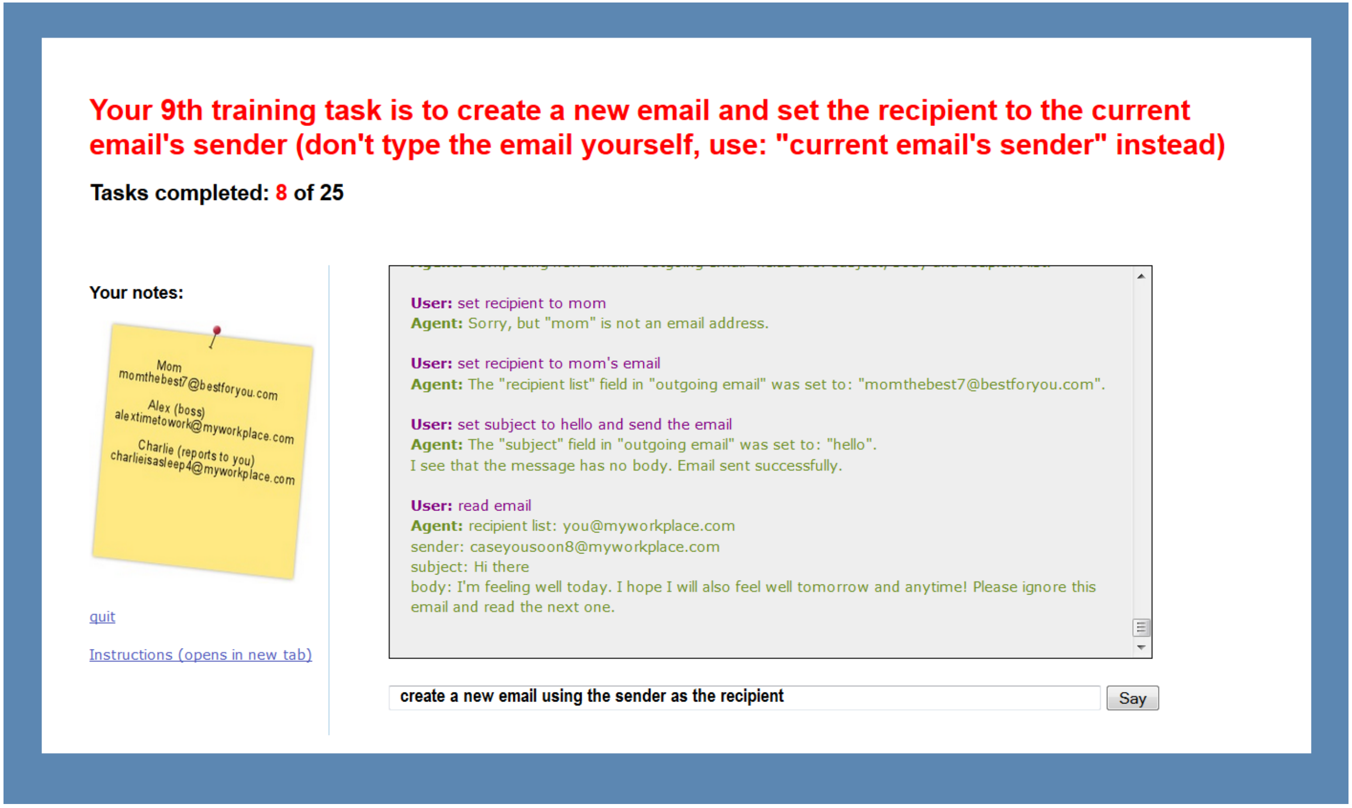
Experimental Setup

We conducted a user study with 123 subjects from Amazon Mechanical Turk, to evaluate our agent. The set of subjects consisted of 60 (48.8%) females and 63 (51.2%) males. Subjects' ages ranged from 19 to 74, with a mean of 35.1. All subjects were residents of the USA. The subject had to fill out a short demographic questionnaire and sign the consent form. They received instructions and had to answer a short quiz to ensure that they fully understood the experiment. (We will use the terms *users* and *subjects* interchangeably in order to avoid confusion with email subjects.)

The interaction page (see Figure 1) included a training phase consisting of 13 tasks that appeared on the top of the screen. After completing all 13 training tasks, the users were given the main task. The training tasks were designed in a way that anyone who completes all training tasks should be able to complete the main task. In the main task, users were requested to read each incoming email and, for each email, to follow the sender's request. There were 12 main tasks, corresponding to requests in 12 emails. An example of such a request is an email sent by Charlie with the following body: "Please email Alex saying that I'm on my way". The user responds to this task by commanding our agent, in natural language, to compose a new email, to set the recipient to Alex's email address (which appears on the left of the screen) and to include a text body in the email stating that 'Charlie is on his way'. Other emails included requests to say something to someone, forward or reply to the given email. The users were instructed to include the same subject both when replying and forwarding an email and include the same body when forwarding an email. None of these tasks *required* the user to teach LIA new commands; however, as we will show, users taught the system new commands that they believed would be useful for future tasks. Users could quit the experiment at any time using the "quit" button. The appendix presents a sample user interaction with LIA that includes teaching new commands.

After completing all tasks or clicking the "quit" button, subjects received an ending questionnaire asking about their acquaintance with programming. The subjects could chose one of the following: None; Very little; Some background from high-school; Some background from college/university; Bachelors (or other degree) with a major or minor in software, electrical or computer engineering or similar; and Significant knowledge, but mostly from other

Figure 1: A screen-shot of a subject interacting with LIA. The training task is shown on top (in red) and communication with LIA is performed using the text pane on the bottom. The previous commands appear above (in purple) along with LIA's responses (in green). The subject's notes (email addresses required for the experiment) appear on the left.



Your 9th training task is to create a new email and set the recipient to the current email's sender (don't type the email yourself, use: "current email's sender" instead)

Tasks completed: 8 of 25

Your notes:

Mom
momthebest7@bestforyou.com
Alex (boss)
alextimetowork@myworkplace.com
Charlie (reports to you)
charlieisasleep4@myworkplace.com

User: set recipient to mom
Agent: Sorry, but "mom" is not an email address.

User: set recipient to mom's email
Agent: The "recipient list" field in "outgoing email" was set to: "momthebest7@bestforyou.com".

User: set subject to hello and send the email
Agent: The "subject" field in "outgoing email" was set to: "hello".
I see that the message has no body. Email sent successfully.

User: read email
Agent: recipient list: you@myworkplace.com
sender: caseyousoon8@myworkplace.com
subject: Hi there
body: I'm feeling well today. I hope I will also feel well tomorrow and anytime! Please ignore this email and read the next one.

create a new email using the sender as the recipient

sources. We assigned numbers to each of the options, resulting in a numeric measure of acquaintance with programming from 1 to 6. (Significant knowledge from other sources was assigned the value 5 while a bachelors in computer science was assigned the value 6.) Subjects were also asked to assign their level of agreement with each of the following sentences on a 7-point Likert scale (Hinkin 1998) (strongly disagree, disagree, slightly disagree, neither agree nor disagree, slightly agree, agree, strongly agree): (1) The computerized agent was smart, (2) The computerized agent understood me, and (3) I would like to have such an agent on my mobile phone.

Results

We evaluate the performance of LIA on the following dimensions: whether new commands were taught, usage of taught commands, completion rate of all tasks, the success rate of the parser and execution, and user rating as obtained from the Likert scale questionnaire. Table 4 summarizes of our results.

LIA was very successful in terms of learning new commands, with all but 4 subjects (among the 50 who completed all tasks) teaching it new commands and reusing them. On average, each subject taught LIA 2.32 commands. We define the *gain from taught commands* as the number of times each taught command was used, multiplied by the number of subcommands executed by this new command. We do not subtract one, since users often define a short new command to replace a single, much more complex, command. This average gain for subjects who completed all tasks was 52.18 commands. Multiplying the gain by 14 seconds – the average time per command – results in an average savings of slightly over 12 minutes per subject. This implies that the subjects saved 39.7% of their time on the main task by

Criterion	Result
Average number of taught commands	2.3
Avg. subcommands gained by taught commands	52.18
Average task completion time	30:28
Average time saved by taught commands	12:06
Percent of time saved by taught commands	39.7%
Completion rate	41%
Parse failure	15.4%
Execution error	5.4%
"The computerized agent was smart" (1-7)	5
"The computerized agent understood me" (1-7)	5.5
"I would like such an agent on my phone" (1-7)	4.6

Table 4: Summary of results.

teaching LIA new commands.

Unfortunately, only 50 subjects (41%) completed all tasks. However, a large percent of those who did not complete all tasks did not seem to struggle with giving commands to LIA. Many of them interacted with LIA only for a few minutes. In fact 65% of the subjects who completed the training, completed all tasks, and 83% of the subjects who completed at least a single task from the 12 main tasks, completed all the tasks. Still, unfortunately, 28.5% of the subjects did not complete all tasks despite spending more than 14:34 minutes on them.[1] The average completion time if we exclude pauses of over a minute (i.e. if the user didn't write anything for over a minute, it is counted just as one minute), is 30:28 (the raw average completion time was: 34:36).

As for the success rate of the parsing and execution, out of a total of $12,654$ commands, 15.4% ($1,952$) were parsed to an unknown command (which in most cases is a parse failure), and an additional 5.4% (679 commands) resulted in an execution error (e.g. trying to change an immutable field or setting an email to a non-email value). When considering only those who completed the tasks, this error rate drops to 9.9% (659 out of $6,649$) of unknown commands, and 4.9% of execution errors. This improvement is expected, since those who completed all tasks interacted longer with LIA and thus were more likely to provide commands that LIA understood. Furthermore, the subjects who provided commands that LIA could not execute were more likely to drop out early. In addition to unknown commands and unsuccessful executions, there were also undesired executions in which LIA executed an action that differed from the subject's intent. This problem can be caused either by a parse error (this happened several times when a subject wanted to set a to b's value, but said set a as b, and, if both a and b were mutable fields, b was set to a's value, due to the more common meaning of the token "as") or human error (e.g. "set recipient to inbox email's recipient", where the subject really ment to use the *sender* of the email in the inbox). In future work we intend to develop methods to measure and correct these types of errors.

As for the Likert scale questions (among those who completed all tasks), the subjects seemed to slightly agree that LIA was smart, giving it exactly 5 on average (slightly agree); the subjects gave LIA 5.5 with respect to whether they thought that LIA understood them (between slightly agree and agree). The subjects gave LIA a score of 4.6 with regards to whether they would like an agent like that on their mobile phone (between neither agree nor disagree, and slightly agree). However, this final question may have been ambiguous because the subjects interacted with LIA using text whereas we intend to use speech commands for mobile phones.

Most commands taught by users were very similar; most subjects taught the system to reply to an email and forward an email. Furthermore, users had a clear preference on the order of subcommands that compose a taught command, even when this order had no impact on the results (such as in which order to set the subject, the body and the recipient of an email). This result suggests that users have similar expectations of commands, which should simplify generalizing commands across users. Nevertheless, we found some unique and interesting commands, such as commands that set only the recipient to be a specific contact's email, and semi-recursive commands. For example, one user first taught "forward," and then taught "forward to charlie," which executes forward, sets the recipient to charlie, then sends the email. LIA generalized this second command so that it could be used with any contact, not just charlie. Another interesting command was taught by a user who kept typing "read" instead of "read email." After making this mistake several times, the user taught LIA to execute "read email" every time it received the command "read."

Interestingly, there was only a weak correlation (0.1) between the level of acquaintance with programming and the number of commands gained by reusing programmed commands. This result is encouraging because it suggests that teaching our agent with natural language instructions does not require programming knowledge. Even the 6 subjects who reported no programming knowledge gained an average of 43.3 commands from teaching, and only one of them did not take advantage of LIA's teachability. There was a stronger correlation between acquaintance with programming and a user's task completion time (-0.31); however, this correlation may be due to other differences between programmers and non-programmers, such as typing speed.

Conclusions and Future Work

In this paper we present our Learning by Instruction Agent (LIA), which using CCG parsing, lambda calculus, and our novel lexicon induction method, is able to learn by instructions given in natural language. LIA receives step by step natural language instructions from users, on how to implement a command (such as forwarding an email to a specific contact), and is able to generalize and later execute this command using different parameters (e.g. to forward a different email to a different contact). We show that with very little training (approximately 10 minutes), and with little or no programming knowledge, many subjects were able to interact with LIA and teach it new commands resulting in a significant time gain of nearly 40% of the time required to complete all tasks.

We suggest that although our current system is only a partial, initial implementation of an instructable agent, our approach can serve as a template for more elaborate systems. If instructable agents of this form could be made widely available to all users of mobile devices, the collective set of commands and natural language phrasings that could be learned by instruction would quickly become quite extensive, changing the nature of mobile devices from systems that can perform only commands built in by their developers, into devices that can be instructed (programmed) by millions of users.

There are significant opportunities for future work. The most obvious direction is supporting more sensors and effectors, both physical and cyber, such as, calendar, caller ID, SMS, GPS and social networks. We intend to deploy

---

[1] 14:34 minutes is the time it took the fastest subject to complete all tasks.

the instructable agent to a community of users and have the users collaboratively define new commands. Deploying LIA also raises questions such as generalizability, and stability. LIA and/or its users will need to have a method to evaluate new commands taught, and decide which should be elevated to the collective knowledge of the agent used by all users and which should remain with the specific user who taught it (since some commands may be personal, confusing, non intuitive and perhaps even adversary). Another direction for future work is to develop a method which will identify whether a command has actually completed what the user intended to or not (this can be done by using similarity to consecutive commands, or attempts to undo). Using this data, the agent can improve its accuracy, both by updating its feature weights and by learning new lexicon entries.

In this current work, we focused solely on procedural execution, i.e. commands that translate to a list of other known commands which are serially executed (with support of arguments and variables). In future work we intend to provide support also to more complex instructions including if-then clauses, which may be interpreted as rules (e.g. "if I receive an email during a meeting, and it is related to the meeting, then notify me immediately").

Acknowledgments

This work was supported in part by Yahoo! through CMU's InMind project. Samsung GRO 2015 grant #A017479 and DARPA under contract number FA8750-13-2-0005.

References

Allen, J.; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. Plow: A collaborative task learning agent. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 1514. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Biermann, A. W. 1983. *Natural language programming*. Springer.

Billard, A.; Calinon, S.; Dillmann, R.; and Schaal, S. 2008. Robot programming by demonstration. In *Springer handbook of robotics*. Springer. 1371–1394.

Calinon, S.; Guenter, F.; and Billard, A. 2007. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 37(2):286–298.

Chen, D. L., and Mooney, R. J. 2008. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, 128–135. ACM.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. *San Francisco, CA* 859–865.

Haas, N., and Hendrix, G. G. 1980. An approach to acquiring and applying knowledge. In *AAAI*, 235–239.

Harnad, S. 1990. The symbol grounding problem. *Physica D: Nonlinear Phenomena* 42(1):335–346.

Hinkin, T. R. 1998. A brief tutorial on the development of measures for use in survey questionnaires. *Organizational research methods* 1(1):104–121.

Koedinger, K. R.; Aleven, V.; Heffernan, N.; McLaren, B.; and Hockenberry, M. 2004. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Intelligent Tutoring Systems*, 162–174. Springer.

Kuklinski, K.; Fischer, K.; Marhenke, I.; Kirstein, F.; Solvason, D.; Kruger, N.; Savarimuthu, T. R.; et al. 2014. Teleoperation for learning by demonstration: Data glove versus object manipulation for intuitive robot control. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2014 6th International Congress on*, 346–351. IEEE.

Le, V.; Gulwani, S.; and Su, Z. 2013. Smartsynth: Synthesizing smartphone automation scripts from natural language. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, 193–206. ACM.

Myers, B. A.; Cypher, A.; Maulsby, D.; Smith, D. C.; and Shneiderman, B. 1991. Demonstrational interfaces: Coming soon? In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 393–396. ACM.

Nakaoka, S.; Nakazawa, A.; Kanehiro, F.; Kaneko, K.; Morisawa, M.; Hirukawa, H.; and Ikeuchi, K. 2007. Learning from observation paradigm: Leg task models for enabling a biped humanoid robot to imitate human dances. *The International Journal of Robotics Research* 26(8):829–844.

Quirk, C.; Mooney, R.; and Galley, M. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, 878–888.

Reed, A. 2010. *Creating Interactive Fiction with Inform 7*. Cengage Learning.

Steedman, M., and Baldridge, J. 2011. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar* 181–224.

Thomason, J.; Zhang, S.; Mooney, R.; and Stone, P. 2015. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the Twenty-Fourth international joint conference on Artificial Intelligence (IJCAI)*.

Zettlemoyer, L. S., and Collins, M. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*.

Appendix

Following is a sample user interaction with LIA that includes teaching new commands:

User: go

Agent: Sorry, but I don't understand. Would you like to teach me what to do in this case (either say yes or simply ignore this question)?

User: yes

Agent: Great! When you say, for example: "go", what shall I do first?

User: move to next email and read it

Agent: Set to next incoming email successfully. sender: caseyousoon8@myworkplace.com subject: Another email body: I felt like sending you another email. Please ignore this email too and read the next one. What shall I do next (when executing: "go")?

User: end

Agent: I'm currently learning the new command ("go"). I'm trying to generalize to other similar commands, this takes several seconds...

Agent: I now know what to do when you say (for example): "go"!

...

User: go

Agent: Set to next incoming email successfully. recipient list: you@myworkplace.com sender: caseyousoon8@myworkplace.com subject: Charlie body: I asked Charlie to do what you said, but I see that it must come from you. Please forward this email to Charlie.

User: create contact Charlie

Agent: Instance "charlie" (of concept "contact") was created. "contact" fields are: email.

User: set Charlie's email to charlieisasleep4@myworkplace.com

Agent: The "email" field in "charlie" was set to: "charlieisasleep4@myworkplace.com".

...

User:go

Agent: Set to next incoming email successfully. sender: alextimetowork@myworkplace.com subject: Work before parting body: We will all have to work very hard next Monday, Tuesday and Wednesday. Please forward this email to Charlie.

User: forward charlie

Agent: Sorry, but I don't understand. Would you like to teach me what to do in this case?

User: yes

Agent: Great! When you say, for example: "forward charlie", what shall I do first?

User: set recipient list to charlie's email, set subject to current email's subject, set body to current email's

Agent: I see that there is no email being composed. Do you want to compose a new email?

User: create new email, set recipient list to charlie's email, set subject to current email's subject, set body to current email's body, send email

Agent: Composing new email. The "recipient list" field in "outgoing email" was set to: "charlieisasleep4@myworkplace.com". The "subject" field in "outgoing email" was set to: "Work before parting". The "body" field in "outgoing email" was set to: "We will all have to work very hard next Monday, Tuesday and Wednesday. Please forward this email to Charlie.". Email sent successfully. What shall I do next (when executing: "forward charlie")?

User: end

Agent: I now know what to do when you say (for example): "forward charlie"!