Training an Agent to Ground Commands with Reward and Punishment

James MacGlashan[*], Michael Littman[*], Robert Loftin[+], Bei Peng[#], David Roberts[+], Matthew E. Taylor[#]

[*] Computer Science Department, Brown University
[+] North Carolina State University
[#] School of Electrical Engineering and Computer Science, Washington State University

Abstract

As robots and autonomous assistants become more capable, there will be a greater need for humans to easily convey to agents the complex tasks they want them to carry out. Conveying tasks through natural language provides an intuitive interface that does not require any technical expertise, but implementing such an interface requires methods for the agent to learn a grounding of natural language commands. In this work, we demonstrate how high-level task groundings can be learned from a human trainer providing online reward and punishment. Grounding language to high-level tasks for the agent to solve removes the need for the human to specify low-level solution details in their command. Using reward and punishment for training makes the training procedure simple enough to be used by people without technical expertise and also allows a human trainer to immediately correct errors in interpretation that the agent has made. We present preliminary results from a single user training an agent in a simple simulated home environment and show that the agent can quickly learn a grounding of language such that the agent can successfully interpret new commands and execute them in a variety of different environments.

1 Introduction

As robots and autonomous assistants become more capable, there will be a greater need for humans to easily convey to agents the complex tasks they want them to carry out. Conveying tasks through natural language provides an intuitive interface that does not require any technical expertise, but implementing such an interface requires methods for the agent to learn a grounding of natural language commands. In this work, we propose a method for learning task groundings of natural language commands from online reward and punishment given by a human trainer.

While there is existing work on learning command groundings (see Section 5), most of it is focused on natural language commands that specify the actions the agent needs to take. However, we should not have to explicitly tell an autonomous agent how to complete a task; the agent should solve the problem on its own. For example, if a human wants a robot to bring them a red bag, they should simply be able to say "bring me the red bag," instead of detailing how the robot needs to navigate to the bag, pick it up, and navigate back to them. Existing approaches for training natural language command groundings focus on using training data that consists of language paired with example demonstrations. However, depending on how complex an agent is, manually controlling it to provide demonstrations may be inconvenient. For instance, if the agent is a robot, teleoperating the robot to provide it demonstrations may 1) be time consuming and cumbersome without a well developed interface, and 2) require the operator to have sufficient expertise to produce high quality demonstrations. An easier way to train the robot is to reward and punish the robot as it is executing a command it was given, similar to how dogs are trained to follow a command through reward and punishment. A reward and punishment paradigm has the additional advantage that if an agent misunderstands a command, a human can easily correct the agent as the agent is attempting to carry out the command, rather than stopping the agent and then providing a new demonstration and language example.

Our approach to learning natural language command groundings makes use of a generative model of tasks, language, behavior, and trainer feedback. The task, language, and behavior model builds on prior work by MacGlashan *et al.* (2014) in which tasks are defined as factored reward functions and termination conditions of an MDP, language is modeled using IBM Model 2 (Brown et al. 1990; 1993), and behavior is generated from the optimal policy for a given task. We extend this approach to enable learning from reward and punishment, rather than demonstration, by including in the model trainer feedback. To model trainer feedback, we adopt algorithms proposed by Loftin *et al.* (2014) for performing task inference from human feedback (without any language grounding) in which feedback is treated as a discrete value (reward, punishment, neutral) and given by a human trainer according to parameters defining their willingness to reward or punish correct and incorrect action selection. The contribution of our work is to show how these two approaches can be combined in a fully interactive system that enables an agent to learn the task groundings of natural language commands.

In our learning algorithm, an agent is first given a command, which induces a probability distribution over the set of possible tasks in the environment. The agent then plans a solution to the most likely task using an "off-the-shelf" MDP planning algorithm and begins following the policy.

After each action execution, the trainer either rewards, punishes, or provides no feedback to the agent, which is evidence for or against each of the possible tasks. If a different task than the one currently being executed becomes more likely than the current one, then the agent switches to following the policy of the new most likely task. After a task has been completed, the final probability distribution over tasks is used to update the language model so that the agent will better understand future commands.

In this work we present preliminary results from a single user training an agent in a simple simulated home environment and show that the agent can quickly learn to correctly interpret and execute new commands.

2 Background

To represent tasks, we make use of the *Object-oriented Markov Decision Process* (OO-MDP) formalism (Diuk, Cohen, and Littman 2008). OO-MDPs extend the conventional MDP formalism by providing a rich state representation. MDPs are defined by a four-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ is a set of states of the world; $\mathcal{A}$ is a set of actions that the agent can take; $\mathcal{T}$ describes the transition dynamics, which specify the probability of the agent transitioning to each state given the current state and the action taken; and $\mathcal{R}$ is a function specifying the reward received by the agent for each transition. MDPs may also have terminal states that cause action to cease once reached. The goal of planning in an MDP is to find a *policy*—a mapping from states to actions—that maximizes the expected discounted cumulative reward.

An OO-MDP extends the classic MDP formalism by including a set of *object classes*, each defined by a set of attributes, and a set of propositional functions whose parameters are typed to object classes. A state in an OO-MDP is a set of objects that each belong to one of the possible object classes; each object has its own state which is represented as a value assignment to the attributes of its associated object class. The propositional functions defined in the OO-MDP are functions of the state of its object parameters. For example, in a blocks world, the function $\texttt{on}(b_1, b_2)$ would operate on two block objects and return true when the state of b_1 was adjacent and above the state of b_2. Although an OO-MDP state is fully defined by the union of objects, these propositional functions provide additional high-level information about the state. In this work, we use propositional functions to define abstract task definitions. Note that although OO-MDPs provide a rich state representation, any standard MDP planning algorithm can be used for planning in an OO-MDP.

3 Approach

Our learning algorithm uses a generative model consisting of task definitions, language, behavior, and trainer feedback. Figure 1 shows a diagram of this model, with arrows indicating conditional probabilities. The next section will give an overview of each part of the model, and present the overall interactive learning algorithm used to learn the parameters of the language model. Our running example is the simplified home environment shown in Figure 2.

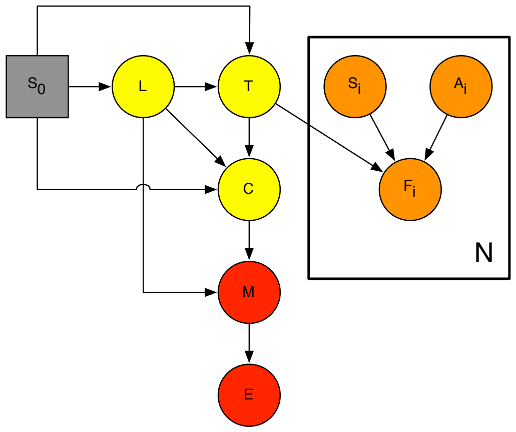


Figure 1: A diagram of the generative model. Arrows indicate conditional probabilities. Variables in the rectangle repeat N times. The gray node is an input initial state of the environment; yellow are task variables; red, language variables; and orange, behavior and trainer feedback variables. L is the random variable for lifted tasks; T, grounded tasks; C, variable binding constraints; M, machine commands; E, English commands; S_i, the ith state; A_i, the ith action; and F_i, the ith feedback.

Task Model

The task model is defined by three components: *lifted task* definitions (L), *grounded task* definitions (T), and *variable binding constraints* (C). The set of possible lifted task definitions are provided by a designer and represent the kinds of tasks an agent could conceivably carry out in an environment. Specifically, each lifted task is a factored reward function defined as a logical expression of OO-MDP propositional functions with the parameters of the propositional functions set to free variables. For example, in the home environment, one lifted task might be to take an object (referred to as "blocks") to some room, which would be expressed as the reward function

$$\mathcal{R}(s, a, s') = \begin{cases} 1 & \text{if } \texttt{blockInRoom}^{s'}(?o, ?r) \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

where s' is the outcome state for the agent after taking action a in state s, $\texttt{blockInRoom}^{s'}$ is a propositional function evaluated in state s', $?o$ is a free variable that can be ground to any movable object in the environment, and $?r$ is a free variable that can be ground to any room in the environment.

If a task is goal directed, then the reward function will also be paired with a similar set of termination conditions.

The probability distribution of the set of lifted tasks is assumed to be uniform over the set of lifted tasks that are *permissible* in an input state. A lifted task is permissible in an input state if there exists at least one object of each object class required by the free variables in the lifted task.

A grounded task (T) is dependent on the lifted task and input state and is an assignment of objects in the input state to each of the free variables in the lifted task. For example,

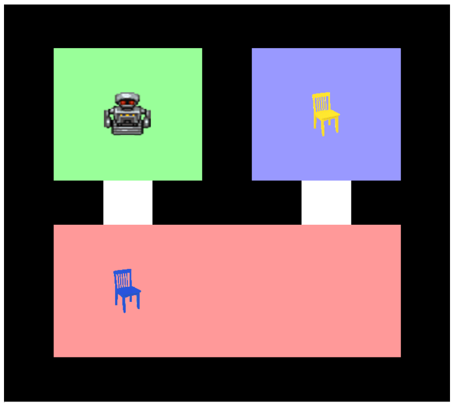


Figure 2: An example three room floor layout with one robot and two chairs. The object reference for the red, green, and blue room are r_1, r_2, and r_3, respectively. The references for the yellow and blue chair are c_1 and c_2, respectively.

given the input state shown in Figure 2, an object assignment to lifted task 1 that represents the task of the robot taking the yellow chair to the red room is $?o := c_1$ and $?r := r_1$. The probability distribution of grounded tasks is uniform over the set of possible variable assignments for the lifted task.

If all an agent could directly infer from a command was the lifted task, it would be impossible for the agent to determine which of the possible grounded tasks for an input state was the intended task. Variable binding constraints (C) are logical expressions that extend a lifted task and which can be inferred from a command, enabling the agent to resolve grounding ambiguity. For example, we can infer from the command "take the yellow chair to the red room," that the grounding of lifted task 1 should satisfy the constraints $\texttt{isYellow}(?o) \wedge \texttt{isRed}(?r)$.

The probability distribution of variable constraints given a grounded task, lifted task, and input state is uniform across the set of possible additional logical constraints that are true in the input state for the given variable assignments in the grounded task. For example, if $?o$ is assigned to c_1 in the grounded task, $\texttt{isYellow}(?o)$, is a possible constraint since it is true in the initial state for c_1, but $\texttt{isBlue}(?o)$ is not.

When the task model is paired with a language model that is dependent on the lifted task and variable binding constraints, Bayesian inference may be used infer the intended grounded task, after which an agent can use standard MDP planning techniques to determine behavior. Specifically, if the language model defines the probability of an English command, given a lifted task and variable binding constraints ($\Pr(e|l,c)$), then the probability of grounded task t is computed by marginalizing over the lifted task (l) and binding constraints (c); specifically,

$$\Pr(t|e,s) \propto \sum_l \Pr(l|s)\Pr(t|s,l)\sum_c \Pr(c|s,l,t)\Pr(e|l,c),$$

where s is the initial input state.

Language Model

The language model we use is IBM Model 2 (Brown et al. 1990; 1993), a statistical machine translation model. In machine translation, the goal is to translate a sentence in a source language (French) into a target language (English). The model consists of translation parameters, giving the probability ($\Pr(f_i|e_j)$) of English word e_j translating to French word f_i; and word alignment parameters, giving the probability ($\Pr(j|i,m,k)$) of the ith word in a French sentence of length k corresponding to the jth word in an English sentence of length m. Given a set of pairs of English and French sentences, these parameters can be learned with Expectation Maximization (Dempster, Laird, and Rubin 1977).

In this work, we use the generative model to translate from an English command e, to a lifted task and variable binding constraints. Because IBM Model 2 is designed to translate between sentences in different languages, the lifted task and variable binding constraints are deterministically transformed into a machine language string m. This transformation iterates over each propositional function in the lifted task and then each propositional function in the variable binding constraints. Each propositional function is turned into a sequence of tokens consisting of the name of the propositional function and the object classes to which each of its parameters are typed. For instance, lifted task 1 and variable binding constraints $\texttt{isYellow}(c_1) \wedge \texttt{isRed}(r_1)$ would become the machine string "blockInRoom block room isYellow block isRed room." Once converted into a machine language string, IBM Model 2 can be used to compute the probability ($\Pr(e|m)$) of an English command e given a machine command m. Translation in the reverse direction is performed using Bayesian inference.

Trainer Feedback Model

The trainer feedback model assumes that a trainer will reward, punish, or do nothing (neutral feedback), in response to the agent taking a correct or incorrect action, with respect to the task they are training. The model contains three parameters: μ^+, μ^-, and ϵ. μ^+ is the probability that the trainer will give neutral feedback when a correct action is taken, and μ^- is the probability that they will give neutral feedback when an incorrect action is taken. ϵ is the probability that the trainer will misinterpret a correct action as incorrect, or *vice-versa*. Given an MDP, an action is assumed to be correct if it is an optimal action for the MDP in the current state, and incorrect otherwise. Therefore, the probability of the agent receiving reward (r), punishment (p), or neutral feedback ($\emptyset$) is defined as

$$\Pr(f=r|s,a,t) = \begin{cases}(1-\epsilon)(1-\mu^+) & \text{if } a \in \pi^t(s)\\ \epsilon(1-\mu^+) & \text{if } a \notin \pi^t(s)\end{cases}$$

$$\Pr(f=p|s,a,t) = \begin{cases}\epsilon(1-\mu^-) & \text{if } a \in \pi^t(s)\\ (1-\epsilon)(1-\mu^-) & \text{if } a \notin \pi^t(s)\end{cases}$$

$$\Pr(f=\emptyset|s,a,t) = \begin{cases}(1-\epsilon)\mu^+ + \epsilon\mu^- & \text{if } a \in \pi^t(s)\\ (1-\epsilon)\mu^- + \epsilon\mu^+ & \text{if } a \notin \pi^t(s)\end{cases},$$

where a is the action the agent took in state s and $\pi^t(s)$ is the set of optimal actions in state s for task t. Given a new

Algorithm 1 Training Command Task Groundings

```
Input: OO-MDP $\setminus\mathcal{R}$ and a set of lifted tasks $L$
  Initialize IBM Model 2 parameters arbitrarily.
  $D \leftarrow \{\}$
  loop forever
    Receive new Environment/State $s_0$
    Receive natural language command $e$
    $\Pr(t) \leftarrow \Pr(t|e, s_o)$
    Plan policy $\pi^t \; \forall t \in T$ where $\Pr(t) > 0$
    $t^* \leftarrow \arg\max_t \Pr(t)$
    $s \leftarrow s_0$
    repeat
      select and take action $a \in \pi^{t*}(s)$
      receive feedback $f$ and next state $s'$
      $\Pr(t) \leftarrow \Pr(t|s, a, f)$
      $t^* \leftarrow \arg\max_t \Pr(t)$
      $s \leftarrow s'$
    until Trainer terminates session
    for $m \in M$ do
      $D \leftarrow D \cup \{(Pr(m|\mathbf{s}, \mathbf{a}, \mathbf{f}, e), m, e)\}$
    end for
    Retrain IBM Model 2 Parameters on $D$
  end loop
```

feedback signal, Bayesian inference can be used to update the belief in each task. Specifically,

$$\Pr(t|s, a, f) \propto \Pr(t)\Pr(f|s, a, t).$$

This model is useful in practice because trainers may give more reward than punishment (*e.g.*, rewarding correct actions and doing nothing for incorrect actions), or *vice-versa*. If the agent knows (or can learn) that the trainer gives more reward or punishment ($\mu^+ \neq \mu^-$), then neutral feedback can provide as much information as explicit feedback (Loftin et al. 2014). Without knowing, however, it is best to set $\mu^+ = \mu^-$, such that neutral feedback is ignored.

Learning Algorithm

Our learning algorithm is shown in Algorithm 1. The algorithm takes as input an OO-MDP definition without the reward function and the set of lifted tasks that the agent can solve. The algorithm begins by initializing the IBM Model 2 parameters arbitrarily and creating an initially empty training dataset D. The algorithm then loops over successive training sessions that begin with a new input state. Each input state may have a different number of objects present, with variable properties, which effectively makes each input state a new environment. The agent is then given a natural language command e that the trainer wants the agent to complete. Using the current IBM Model 2 parameters, the prior over grounded tasks is updated to the posterior given the command. The agent then uses any "off-the-shelf" MDP planning algorithm to find the policy for each task. Next begins a behavior loop in which the agent follows the policy of the most likely grounded task, receives a feedback, updates its belief in each grounded task, and switches to the policy of the new most likely grounded task. This behavior loop continues until the trainer terminates the session. For each possible machine language command m, a training instance is added to dataset D. Each training instance consists of the machine language command (m), the natural language command (e), and a weight that is the posterior probability of m given the initial command e, and the state, action, and feedback sequence observed during the last behavior loop. This probability is computed by marginalizing over the lifted task, grounded task, and binding constraints:

$$Pr(m|\mathbf{s}, \mathbf{a}, \mathbf{f}, e) \propto \sum_{l,t,c} \Pr(l|s)\Pr(t|s, l)\Pr(\mathbf{f}|\mathbf{s}, \mathbf{a}, t) \Pr(c|s, l, t)\Pr(m|l, c)\Pr(e|l, c),$$

where $\Pr(\mathbf{f}|\mathbf{s}, \mathbf{a}, t)$ is the probability of feedback sequence $\mathbf{f}$, which is computed by treating each feedback as independent of the other state-action-feedback triples in the sequence:

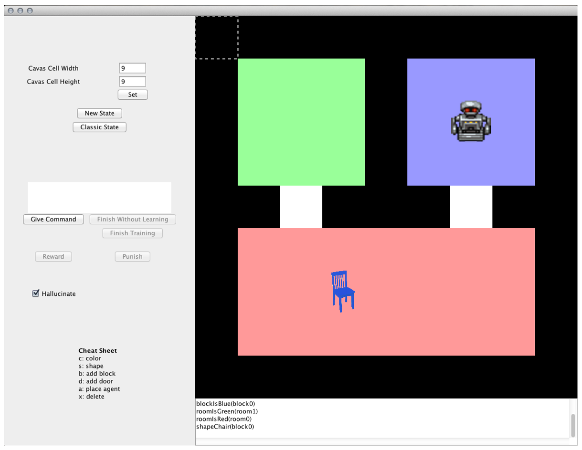
$$\Pr(\mathbf{f}|\mathbf{s}, \mathbf{a}, t) = \prod_i \Pr(f_i|s_i, a_i, t).$$

Finally, the IBM Model 2 parameters are retrained using the updated dataset D. The training algorithm for IBM Model 2 is a weakly supervised Expectation Maximization (EM) algorithm, as in prior work (MacGlashan et al. 2014), similar to the classic EM algorithm for IBM Model 2, except that the contribution of each data instance is weighted, as the intended task must be inferred probabilistically.

4 Experimental Results

To test the model, we performed preliminary experiments with a single trainer (one of the authors) training an agent in the same simplified simulated home environment discussed previously. The OO-MDP representing this domain consists of four object classes: `AGENT`, `ROOM`, `DOOR`, and `BLOCK`. The `AGENT` class is defined by x and y position attributes; the `ROOM` class is defined by attributes defining the bounding box of the room and the color the room is painted; the `DOOR` class is defined by attributes defining its bounding box; and the `BLOCK` class is defined by x and y position attributes, a color attribute, and a shape attribute. The possible color values are red, green, blue, yellow, and purple; the possible shapes are chair, bag, backpack, and basket (though in these experiments only the chair and bag were used). The agent can deterministically move one unit north, south, east, or west and the agent can move blocks by moving into them. Propositional functions in the OO-MDP include propositional functions for checking if a block is a certain shape (e.g., `shapeChair(BLOCK)`), checking if a block is a certain color (e.g., `isRed(BLOCK)`, checking if a room is painted a certain color, checking if the agent is in a room (`agentInRoom(AGENT, ROOM)`), and checking if a block is in a room (`blockInRoom(BLOCK, ROOM)`).

The possible lifted tasks given to the agent were a task for taking a block to a room, as previously defined in Equation 1, as well as a task for going to a room, which is similar in form except it uses the `agentInRoom` propositional function. Because these tasks are goal directed, the corresponding goal states were also terminating states. As this domain is deterministic, uniform cost search was used for planning. If the agent reached a terminating state for the



Cavas Cell Width: 9
Cavas Cell Height: 9
☑ Hallucinate

Cheat Sheet
c: color
s: shape
b: add block
d: add door
a: place agent
x: delete

```
blockIsBlue(block0)
roomIsGreen(room1)
roomIsRed(room0)
shapeChair(block0)
```

Figure 3: The GUI used to create environments, give commands, and train the agent.

task it was completing, then it was forced to always take a special "noop" action that did not change the state.

The training interface, shown in Figure 3, lets the trainer create custom environments, give commands, and then interactively reward and punish the agent. A training session can be ended at any time with the choice of adding it to the language model's training dataset or ignoring it.

We chose a training sequence of environments/initial states and commands to iteratively teach the agent about a different property of the commands. The sequence of training environments are shown in Figures 4a to 4f (4g is only used for testing after training in the former environments), with environment 4a being used twice in a row and the rest only used once.

The sequence of commands and the associated environment in which they were given are shown in Table 1. Note that in the first four examples, the agent is told to go to a different colored room with no other objects in the environment. This initial set of training commands is to train the agent on the meaning of the room color words. In the following three commands, a single movable object is present, which the agent is asked to take to a room. The first time, it's a blue chair, then a purple bag, and then a yellow bag. The goal of this training sequence is that the agent should have been given enough examples to disambiguate which words refer to room color, block color, or block shape.

Table 1: Sequence of training commands and environments.

#	Command Given	Associated Environment
1.	go to the green room	4a
2.	go to the red room	4a
3.	go to the blue room	4b
4.	go to the yellow room	4c
5.	take the blue chair to the yellow room	4d
6.	take the purple bag to the blue room	4e
7.	take the yellow bag to the red room	4f

During each training session, the trainer employed a feedback strategy of giving reward and punishment intermittently for correct and incorrect action selection. The trainer feedback strategy parameters were set to values that reflect this strategy ($\mu^+ = 0.7, \mu^- = 0.7$, and $\epsilon = 0.1$). It is worth noting that during training, the agent correctly interpreted the command without any required reward or punishment (though it was given anyway) for training commands 2, 5, 6, and 7. For command 2, it was most likely luck that the agent interpreted the command correctly since it had such limited experience. Although the objects to move in instance 5, 6, and 7 were novel, there was only one possible block to move, which meant that to interpret the command correctly, all the agent needed to know was that it was a block moving task and which color room to take the block, which it should have been able to determine from its previous experiences.

To test if the agent learned to properly associate words with the relevant semantic components after the training period, a single testing environment, shown in Figure 4g, was used and after each test command in this environment, the training session was completed without updating the language model so that any new data did not contaminate the results. Note that this environment is different from all former environments and includes a block shape and color pairs not previously seen. Specifically, this environment included a purple chair and a blue bag. If the language model associated words with the correct semantic components, however, then the agent would be able to properly interpret the commands. The test commands given are shown in Table 2. The

Table 2: Sequence of test commands, in environment 4g.

#	Test commands
1	take the purple chair to the green room
2	take the blue bag to the yellow room
3	go to the green room
4	go to the red room
5	go to the blue room

first two test commands test whether the agent was able to properly disambiguate which block it was being requested to move, even though the agent had never been asked to move that specific kind of block. The last three test sentences were designed to test whether the agent still understood commands to go to rooms, even in the presence of novel objects. The agent correctly interpreted the test command in all cases.

Another way to gain insight into what the agent learned is to examine the translation parameters of IBM Model 2 after training is completed. Although there is not space to list all the parameters, we provide some interesting examples here. The machine language word "agentInRoom" was most likely to generate the English word "go" (with probability 0.51); "shapeBag" was most likely to generate "bag" (probability 0.32); "roomIsRed" was most likely to generate "red" (probability 0.33); and "blockInRoom" was most likely to generate the word "take" (probability 0.33).

As a final test to demonstrate the utility of modeling different feedback strategies, this same experiment was rerun, except the trainer was only allowed to punish the agent. To

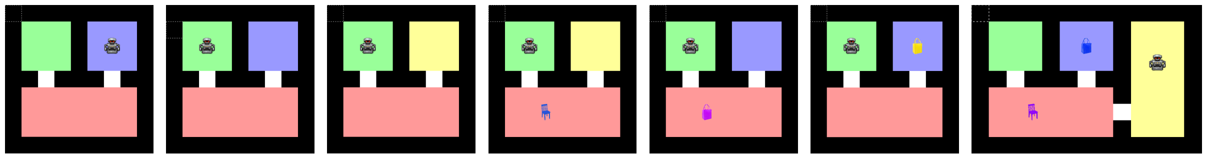

(a) Train 1-2 (b) Train 3 (c) Train 4 (d) Train 5 (e) Train 6 (f) Train 7 (g) Test 1-5

Figure 4: The training evnrionments (a-f) and test environment (g).

reflect this change in strategy, the the trainer feedback model parameters were changed accordingly[1] ($\mu^+ = 0.7, \mu^- = 0.05$). During training, the agent only incorrectly interpreted commands one and three, which means for the remaining training sessions, the trainer provided no explicit feedback at all. Despite this lack of explicit feedback, when coupled with the feedback strategy model parameters, the agent was still able to learn the command groundings. On the test commands, the agent correctly interpreted test commands 1-4. The agent did misinterpret test command 5, but switched to the correct behavior after a single punishment.

5 Related Work

The most similar work to ours is work that learns command groundings from demonstrations in the world, rather than using annotated text. For example, previous work that investigates grounding commands from demonstration included work on giving navigational instructions (Vogel and Jurafsky 2010; Chen and Mooney 2011; Grubb et al. 2011), giving action commands to a robot (Duvallet, Kollar, and Stentz 2013; Tellex et al. 2011; 2014), controlling software (Branavan et al. 2009; Branavan, Zettlemoyer, and Barzilay 2010) and playing games (Goldwasser and Roth 2011; Branavan, Silver, and Barzilay 2011). Two critical differences between this body of work and ours is (1) that our work aims to ground language to task descriptions that do not require the human to tell the agent how to complete the task and (2) our learning algorithm grounds language from online reward and punishment delivered by a human.

Two related approaches to training an agent with human reward and punishment are TAMER (Knox and Stone 2009) and COBOT (Isbell et al. 2001). In these algorithms, human feedback is modeled as a reward function which is then used in a typical reinforcement learning algorithm. One disadvantage to this approach is that it cannot make use of different training strategies employed by the trainer; additionally, the modeled reward function is highly dependent on each state in the state space, which does not capture an overall task definition that can easily be connected with language.

Other related work focuses on how to learn the behavior, rather than identify the task that needs to be completed. Rybski *et al.* (2007) propose a method to train a robot to complete novel tasks from demonstrations that incorporate language with *a priori* grounded utterances. Tenorio-Gonzalez *et al.* (2010) use language to shape an already known reward function. Hewlett *et al.* (2010) learn the behavior of specified verb phrases as abstract finite state machines. Future work could incorporate such approaches to reduce the computational complexity of planning.

6 Conclusions

We proposed a learning algorithm for grounding natural language commands to high-level task definitions from online human-delivered reward and punishment. Grounding commands to high-level task definitions is especially useful because it enables the human to convey tasks to an agent without having to tell the agent how to carry it out. As a result, the same command can be given in a variety of different states/environments and the agent will autonomously determine how to carry out the task in each of them. Training with reward and punishment has the advantage that it does not require technical expertise in manually controlling the agent to provide demonstrations.

We presented preliminary results of a single user interactively training an agent in a simplified simulated home environment and demonstrated that after only a few examples the agent was able to correctly interpret novel commands. In the future, we will perform a user study to assess how well the system handles the kind of variable language and feedback that would be provided by novice users who do not understand the underlying system.

A limitation of our approach is that the agent must plan for every possible task that could be performed in the environment. As environments get more complex and more tasks are possible, this will become intractable. In the future, we would like to explore sampling methods to reduce the space of tasks considered and couple that with “scaffolding” where initially training with simple environments allows the agent to easily learn word groundings, and therefore allows the agent to focus on the most likely task groundings for a command in more complex environments.

In this work, we focused on providing the agent no background language knowledge, since being able to learn without any background language knowledge reduces the amount of work required by a designer. However, in practice, background language knowledge could be incorporated to further accelerate learning. For example, a dictionary could be used to generalize the grounding of one observed word to its synonyms. Additionally, if certain words were known to correspond to colors, then they could similarly have their translation probabilities biased toward being generated by color propositional functions.

[1]Alternatively, these parameters could be learned online using the I-SABL algorithm (Loftin et al. 2014).

7 Acknowledgements

This work was supported in part by NSF IIS-1149917 and NSF IIS-1319412.

References

Branavan, S. R. K.; Chen, H.; Zettlemoyer, L. S.; and Barzilay, R. 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.

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

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

Brown, P. F.; Cocke, J.; Pietra, S. A. D.; Pietra, V. J. D.; Jelinek, F.; Lafferty, J. D.; Mercer, R. L.; and Roossin, P. S. 1990. A statistical approach to machine translation. *Comput. Linguist.* 16(2):79–85.

Brown, P. F.; Pietra, V. J. D.; Pietra, S. A. D.; and Mercer, R. L. 1993. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.* 19(2):263–311.

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

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

Diuk, C.; Cohen, A.; and Littman, M. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*.

Duvallet, F.; Kollar, T.; and Stentz, A. T. 2013. Imitation learning for natural language direction following through unknown environments. In *IEEE International Conference on Robotics and Automation (ICRA).*

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

Grubb, A.; Duvallet, F.; Tellex, S.; Kollar, T.; Roy, N.; Stentz, A.; and Bagnel, J. A. 2011. Imitation learning for natural language direction following. In *Proceedings of the ICML Workshop on New Developments in Imitation Learning.*

Hewlett, D.; Walsh, T. J.; and Cohen, P. R. 2010. Teaching and executing verb phrases. In *Proceedings of the First Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-Epirob-11).*

Isbell, C.L., J.; Shelton, C.; Kearns, M.; Singh, S.; and Stone, P. 2001. A social reinforcement learning agent. 377 – 384.

Knox, W. B., and Stone, P. 2009. Interactively shaping agents via human reinforcement: The tamer framework. 9 – 16.

Loftin, R.; MacGlashan, J.; Littman, M.; Taylor, M.; and Roberts, D. 2014. A strategy-aware technique for learning behaviors from discrete human feedback. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014).*

MacGlashan, J.; Babeş-Vroman, M.; desJardins, M.; Littman, M.; Muresan, S.; and Squire, S. 2014. Translating english to reward functions. Technical Report CS14-01, Computer Science Department, Brown University.

Rybski, P. E.; Yoon, K.; Stolarz, J.; and Veloso, M. M. 2007. Interactive robot task training through dialog and demonstration. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, 49–56. IEEE.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M.; Banerjee, A. G.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificical Intelligence.*

Tellex, S.; Thaker, P.; Joseph, J.; and Roy, N. 2014. Learning perceptually grounded word meanings from unaligned parallel data. *Machine Learning* 94(2):205–232.

Tenorio-Gonzalez, A. C.; Morales, E. F.; and Villaseñor-Pineda, L. 2010. Dynamic reward shaping: training a robot by voice. In *Advances in Artificial Intelligence–IBERAMIA 2010*. 483–492.

Vogel, A., and Jurafsky, D. 2010. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL 2010).*