Mach Learn (2014) 94:205–232
DOI 10.1007/s10994-013-5407-y

Learning from natural instructions

Dan Goldwasser · Dan Roth

Received: 15 May 2012 / Accepted: 8 August 2013 / Published online: 18 September 2013

Abstract Machine learning is traditionally formalized and investigated as the study of learning concepts and decision functions from labeled examples, requiring a representation that encodes information about the domain of the decision function to be learned. We are interested in providing a way for a human teacher to interact with an automated learner using *natural instructions*, thus allowing the teacher to communicate the relevant domain expertise to the learner without necessarily knowing anything about the internal representations used in the learning process.

In this paper we suggest to view the process of learning a decision function as a natural language *lesson interpretation problem*, as opposed to learning from labeled examples. This view of machine learning is motivated by human learning processes, in which the learner is given a lesson describing the target concept directly and a few instances exemplifying it. We introduce a learning algorithm for the *lesson interpretation problem* that receives feedback from its performance on the final task, while learning jointly (1) how to interpret the lesson and (2) how to use this interpretation to do well on the final task. traditional machine learning by focusing on supplying the learner only with information that can be provided by a task expert.

We evaluate our approach by applying it to the rules of the solitaire card game. We show that our learning approach can eventually use natural language instructions to learn the target concept and play the game legally. Furthermore, we show that the learned semantic interpreter also generalizes to previously unseen instructions.

Keywords Semantic interpretation · Indirect supervision · Structure prediction

Editors: Antoine Bordes, Ronan Collobert, and Luke Zettlemoyer.

D. Goldwasser (✉) · D. Roth
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, USA
e-mail: goldwas1@illinois.edu

D. Roth
e-mail: danr@illinois.edu

1 Introduction

Machine learning has traditionally focused on learning concepts in a supervised setting: given a set of labeled examples, the learner constructs a decision function generalizing over the observed training data. While this approach has been tremendously successful for many learning domains, it carries an inherent drawback: the learner can only be as good as the data it is given. Learning therefore depends on annotating considerable amounts of training data, an expensive and time consuming process. Furthermore, successful learning often depends on machine learning expertise required for designing and calibrating the learning environment.

In this work we take a first step towards alleviating some of this difficulty by suggesting a different kind of learning protocol: Learning from Natural Instructions (LNI). This protocol draws motivation from human learning processes, in which the learner is given a "knowledge injection", in the form of Natural Language (NL) instructions describing a high level target concept and a few specific examples to validate the correct understanding of the lesson. Under these definitions, *learning* is viewed as the process of converting a natural language representation of the target concept into a machine language representation that can be understood by an automated agent. The following example shows such a pair: a lesson and its corresponding logical representation. This paper analyzes the process of teaching rules of card games, the example therefore describes a precondition for a Solitaire card game *move*.

Example 1 Solitaire card game NL instruction and logical form output
"You can move any top card to a free cell if it is empty"
$Move(a_1, a_2) \leftarrow card(a_1) \wedge top(a_1, x_1) \wedge freecell(a_2) \wedge empty(a_2)$

Supervised learning, an inductive process by nature, generalizes over the labeled examples. This contrasts with our approach, which attempts to learn the correct hypothesis directly by interpreting NL instructions, rather than by inducing it from examples. This conceptual difference between the approaches makes each one preferable in different situations. For example, to effectively use LNI, a succinct definition of the target concept (i.e., the classification rule) is required. Capturing the intricacies required for defining an object detector in images using natural language might not be possible, and is better approached using traditional example-based learning; however, defining the relations between objects in a scene can be done concisely using natural instructions, making use of a vocabulary, some of which might have been learned using example based learning. More broadly, learning from natural instructions is concerned with communicating symbolic knowledge between a human teacher and an automated agent. This approach carries with it an immediate advantage, as it allows the system designer to focus on task-related expertise. To ensure this advantage, we focus here on natural instructions, describing the target concept in natural language.

While the promise of this approach is clear, successful learning in these settings depends on correctly communicating relevant knowledge to the learning system. Unfortunately this proves to be a non trivial task: allowing human users to communicate effectively with computer systems in a natural manner is one of the longest standing goals of artificial intelligence. This problem, often referred to as *semantic parsing*, is typically framed as a natural language interpretation task, mapping between natural language input and a formal meaning interpretation expressed in a logical language understandable by the target computer system. Current approaches employ machine learning techniques to construct a semantic parser. The learning algorithm is given a set of input sentences and their corresponding meaning representations and learns a statistical semantic parser—a set of semantic parsing rules mapping lexical items and syntactic patterns to their meaning representation and a score associated with each rule. Given a sentence, the semantic parsing rules are applied recursively to derive the most probable meaning representation. We refer the reader to Sect. 4, where we describe this process in more details. Since semantic interpretation is limited to syntactic patterns identified in the training data, the learning algorithm requires considerable amounts of annotated data to account for the syntactic variations associated with the meaning representation. Annotating sentences with their corresponding logical meaning representation is a difficult, time consuming task. The supervision effort required for learning is a major challenge in scaling semantic parsing.

The difficulty of constructing a semantic parser presents us with a major obstacle, since LNI shifts the weight from learning the target concept (e.g., a card game rule), to learning a semantic parser for natural language instructions describing the target concept. Taking a supervised learning approach for constructing a semantic parser would result in an equally hard learning problem (if not a harder) than the one we hoped to avoid. Our learning framework evades this difficulty by making the connection between the two learning tasks explicit. In our settings, the learner has the ability to test its understanding over a handful of labeled examples (e.g., solitaire card game moves). This small set of examples is insufficient for constructing the target hypothesis, but it allows the learner to reject incorrect lesson interpretations. We exploit this property to provide feedback to the semantic interpretation learning process, and base the learning algorithm for semantic parsing on this feedback.

In this paper we examine learning in such settings, where the prediction of the learning algorithm is executed by a computer program resulting with a response or observable action in the target domain. We propose a response driven learning framework that is capable of converting feedback to this action to supervision, and use it to further learn a better definition of the target. This type of supervision is very natural in many situations and requires no machine learning expertise and thus can be supplied by any user.

This paper extends the work previously presented in Goldwasser and Roth (2013); Clarke et al. (2010) by suggesting extensions to the learning algorithm and providing new experimental results. Continuing with Example 1, the response is generated by letting the agent *play game scenarios* using the classification rule it generated. These moves are done against a game API that could accept the move as a legal one, or reject it—thus providing a binary feedback signal. We consider scenarios where the feedback is provided as a binary signal, correct $+1$ or incorrect -1. This weaker form of supervision poses a challenge to conventional learning methods: semantic parsing is in essence a structured prediction problem requiring supervision for a set of interdependent decisions (the predicates in the representation and the relations between them), while the provided supervision is binary, only indicating the correctness of a generated meaning representation. To bridge this difference, we propose a novel learning algorithm suited to the response driven setting, that can make use of this weak supervision signal to improve its semantic parsing model. We discuss the learning algorithm in detail in Sect. 3 of this paper.

Furthermore, to account for the many syntactic variations associated with the output meaning representation, we propose a new model for semantic parsing that allows us to learn effectively. Current semantic parsing approaches extract parsing rules mapping natural language sentences to their logical meaning representation, restricting possible interpretations only to previously seen syntactic patterns. We replace this rigid inference process induced by the learned parsing rules with a flexible framework. We model semantic interpretation as a sequence of interdependent decisions, each mapping a text span to a logical symbol and use syntactic information to determine how the meaning of these logical fragments should be composed. We frame this process as an Integer Linear Programming (ILP) problem, a powerful and flexible inference framework that allows us to inject relevant domain knowledge, such as specific domain semantics that restrict the space of possible interpretations, into the inference process. We explain this terminology and discuss the interpretation process in detail in Sect. 4.

We test our approach in an actionable setting, in which the learner can evaluate its hypothesis by taking actions in a (simulated) world environment. We begin by evaluating our learning algorithm and unique interpretation process on the well studied Geoquery domain (Zelle and Mooney 1996; Tang and Mooney 2001), a database consisting of U.S. geographical information and natural language questions. Although the textual input in this domain does not describe classification rules, but rather database queries, it allows us to evaluate our learning algorithm and its properties and, in addition, to compare our approach to existing work. We then evaluate our overall LNI approach on a set of Solitaire card game rules, in which the lessons describe preconditions on actions. We show that, by applying our protocol, an automated system can be taught the rules required for playing games using natural language instructions.

2 Learning from natural instructions

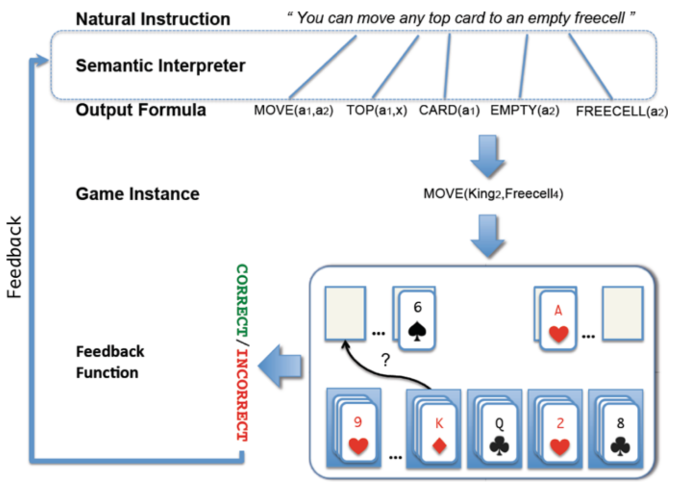
In this section, we give a bird's eye view of our framework and its components. The purpose of our framework is to learn a classification function capturing an observable desired behavior of the learning system. However, unlike traditional machine learning algorithms, our framework does not learn from annotated examples, but rather from natural instructions describing the target concept. The differences between the two frameworks are summarized in Fig. 1.

The learning process aims at improving the ability of the system to understand instructions. The only supervision signal available to the learner is the system's behavior given its interpretation of the instructions, following the intuition that correct behavior corresponds to a correct interpretation of the input instructions. Section 3 describes the learning process in detail.

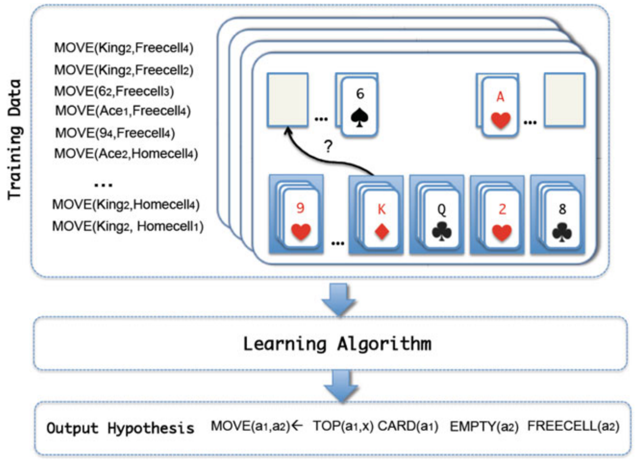
The interpretation process is formulated as a structure prediction task, mapping a complex input object, a natural language instruction, into a complex output structure, a set of interconnected logical entities. This process is briefly defined below in Sect. 2.1, and in more details in Sect. 4.

Our experiments are designed to evaluate the overall LNI approach in two domains. In the card game domain we used the Freecell solitaire card game, taking as input instructions describing the legality of game actions and used the instructions' interpretations to predict which moves are legal given specific game states. After learning terminates, the agent is presented with natural language instructions describing a new rule, and without using any feedback resources the instructions are converted into a logical rule and tested on relevant game data.

Figure 1(a) describes an example of this process. Since our framework is concerned with two learning processes, learning to interpret game instructions describing game rules *and* learning game rules, we consider two evaluation measures. The first evaluates the quality of learning a game rule, by testing the learned system over unseen game examples. The second evaluates quality of the semantic interpretation model by testing its behavior on previously unseen natural language definitions of rules.



(a) Language interpretation setup—from textual input to real world behavior. The learner takes in as input a natural language instruction and utilizes a feedback function to test that the resulting formula was correctly generated from the natural language input. Learning is driven by this feedback, and no feedback is given directly at the level of the intermediate semantic representation. The result of the learning process is both a game move classification function and a semantic interpreter.



(b) Learning in a supervised setting depends completely on labeled game moves. The result of learning is a game moves classification function. In this example we assume an inductive logic programming learner which constructs a first order formula from labeled examples consisting of a game state and a game move.

Fig. 1 LNI vs. Supervised learning. LNI learns a game classification function from interpreting natural language instructions. The result of LNI is both a semantic interpreter and the game rule described by the text

In addition to the card game domain, we take a closer look the semantic interpretation learning process by studying a different domain, a natural language database access domain (Geoquery). In Sect. 5 we provide further details.

2.1 Semantic interpretation

We formulate semantic interpretation as a structured prediction problem, mapping a natural language input sentence (denoted by **x**), to its highest ranking logical interpretation (denoted by **y**). The term *structure* refers to the set of interdependent output decisions (**y**). Since these decisions are typically interrelated, taking these decisions independently may lead to a sub-optimal solution. The preferred approach therefore is to consider them together by optimizing a joint objective function.

In order to correctly parametrize and weigh the possible outputs, the decision relies on an intermediate representation: an alignment between textual fragments and their meaning representation (denoted **h**). This intermediate representation captures the mapping between individual natural language fragments and logical fragments. These concepts are exemplified in Fig. 2, and the notation used to described them is summarized in Table 1.

In our experiments, the input sentences **x** are taken from two domains: natural language Freecell Solitaire game instructions (which describe the legality of possible actions) and geographical database queries. The output formula representation **y** is described using a formal language. We provide further details about it in Sect. 2.2.

The prediction function, mapping a sentence to its corresponding interpretation, is formalized as:

$$\hat{\mathbf{y}} = F_{\mathbf{w}}(\mathbf{x}) = \arg\max_{\mathbf{h}\in\mathcal{H},\mathbf{y}\in\mathcal{Y}} \mathbf{w}^T \Phi(\mathbf{x}, \mathbf{h}, \mathbf{y}) \tag{1}$$

where Φ is a feature function defined over an input sentence **x**, alignment **h** and output **y**. The weight vector **w** contains the model's parameters, whose values are determined by the semantic interpretation learning process.

We refer to the arg max above as the inference problem. Given an input sentence, solving this inference problem based on Φ and **w** is what comprises our interpretation process.

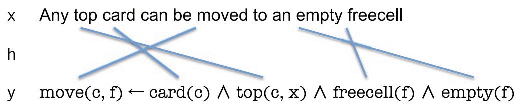

x
Any top card can be moved to an empty freecell
h
y
move(c, f) ← card(c) ∧ top(c, x) ∧ freecell(f) ∧ empty(f)

Fig. 2 A natural language sentence (denoted **x**), an output structure, a logical formula corresponding to the sentence (denoted **y**), and a lexical alignment between the two (denoted **h**)

Table 1 Notation summary

Notation	Explanation	Section
x	Input sentence	Sect. 2.1
y	Output structure	Sect. 2.1
h	Hidden structure	Sect. 2.1
Φ	Feature function	Sect. 2.1
D	Set of domain symbols	Sect. 2.2
`args(p)`	Mapping to the set of arguments of a function `p`	Sect. 2.2

In practice, the semantic parsing decision, mapping a sentence into a logical formula, consists of many smaller decisions. The structure consists of two types of decisions: (1) Lexical decisions, which correspond to lexical alignment decision (encoded via **h**); these determine which logical *symbols* will appear in the output formula based on lexical evidence (2) Compositional decisions, which determine how the symbols should be *composed* into a logical formula that can be evaluated in a given a game state. For example, consider the mapping described in Fig. 2. The mapping between the word "freecell" and the predicate `freecell(f)` is an example of the first decision type. The fact that the function `freecell(f)` and the function `empty(f)` are applied over the same variable is an example of the second type of decisions. Section 4 provides more details about the feature representation and inference procedure used.

In this work we assume that there is a weak conceptual alignment between the set of logical domain symbols used to describe the game and the language used to describe it. This alignment is unknown and imperfect—we do not assume a one-to-one correspondence between words and logical symbols, but rather assume that the entities and relations described by the natural language sentence can be grounded into domain symbols. In general, this assumption underlies all semantic parsing work.

2.2 Target representation

The output of semantic interpretation is a logical formula, grounding the semantics of the input sentence in the target domain (e.g., Solitaire card game, or Geoquery). We use a subset of first order logic consisting of: (1) typed constants (corresponding to specific cards, values, in the Solitaire domain, or states, cities and other geographic concepts, in the Geoquery domain), and (2) functions, which capture relations between domain entities, and properties of entities (e.g., $\texttt{value} : E \rightarrow N$, where $e \in E$ refers to an entity such as a card, and $n \in N$ is an integer). Our formulation of the Solitaire domain is an extended version of the Freecell domain defined in the Planning Domain Definition Language (PDDL), which is used for evaluating automated planning systems. We refer the reader to Zelle and Mooney (1996) for more details about the Geoquery domain.

The Solitaire domain consists of 87 constants, such as cards and their values, colors and suits. A game state is described by assigning truth values to logical functions which map constants to properties (such as locations), the game is described using 14 logical functions. Throughout the paper we denote the set of logical symbols in the domain as **D**. A *game state* contains specific instantiations of the domain symbols, describing the relations between the entities in the state. Given a game state, a logical formula defined over the domain symbols can be evaluated. For example, with regard to the game state described in Fig. 2, the formula $\texttt{freecell}(x_1) \wedge \texttt{top}(x_2, x_1) \wedge \texttt{value}(x_2, 6)$, stating that a card with a value of 6 is on top of at least one freecell, will be evaluated to `true`.

As can be observed in this example, dependency among the values of logical functions are expressed via argument sharing. We use this mechanism to construct meaningful logical formulas from text, that can be evaluated given game states. Our goal is to predict the legality of game actions, these are expressed as horn rules, an implication from an antecedent to a consequent (a single formula referred to as the head). We refer to the predicate corresponding to an action, `move`, as the *head* predicate and we denote its arguments by a_1, a_2. We define `args(p)` to be a function mapping a predicate p to its list of argument variables, and denote by p^i the i-th argument in this list. The Geoquery domain uses a more restricted logical representation language, variable-free first order logic. In this case the meaning of a sentence is constructed using function application and composition operators (e.g., `city(New_York)`).

2.3 Feedback from real world behavior

Observing the behavior resulting from instruction interpretation is the only feedback mechanism available to our learner. We envision a human learning process, in which in addition to high level instructions the learner receives a small number of examples used to clarify these instructions. These examples consist of a game state and a move action whose legality depends on the truth values of functions in the given game state, and the target move described by the instruction. For each concept taught, the learner had access to a handful of positive examples (a legal move in the given game state) and negative examples (an illegal move in the given game state) which were chosen randomly. The feedback is obtained by comparing the outcome of the rule generated by the learned semantic interpreter when applied to these examples, with the true label. The feedback function is a conjunction of these comparisons.

Since our goal is to learn the target concept from instructions rather than from examples, we designed a weak feedback mechanism which cannot be used to learn the target concept directly as the set of examples is too small. Note that although the feedback function can be called many times during the overall learning process, the effort involved in constructing the feedback function remains unchanged. This contrasts with other learning protocols that involve a learner querying an external teacher, such as active learning (Cohn et al. 1994). An active learner intelligently explores the space of possible examples by selectively requesting labels from an external teacher, while in our setting the number of available labeled examples remains a small constant.

A different protocol, in which the learner is allowed to generate its own set of training examples by sampling game moves while actually playing the game would result in a costly feedback function. In other domains, in which the construction of the feedback function does not rely on human annotation effort, this protocol can be applied.

In our setup we assume that there is a strong correspondence between the feedback function and the correctness of the decision rule obtained by interpreting the natural language lesson. This allows our semantic interpretation learning algorithm to use this type of behavioral feedback when constructing a semantic parser. In order to ensure this correspondence, we treat the different move action types of each game independently by assigning each action a natural language instruction set and a dedicated feedback function (i.e., by using a small set of labeled examples for each rule). A considerably more difficult scenario in which all the game rules are learned together without dedicated feedback functions, is left as future work.

In the Geoquery domain, the natural language input consists of geographical queries. These NL queries are converted into a logical formula used to query a geographical database. In order to construct the feedback function we supplied the correct answers to these questions, and compared the results obtained by the queries interpretation. Generating this type of feedback is considerably easier than generating the logical queries which capture the semantics of the NL questions. For example, we paired the query "*what is the capital of Texas?*" with the result *Austin*. A positive feedback will only be generated if the same response will be returned when using the NL query interpretation to query the database.

Throughout the paper we abstract over the implementation details and refer to the feedback mechanism as a binary function $\mathit{Feedback} : \mathcal{Y} \rightarrow \{+1, -1\}$, informing the learner whether a predicted logical form $\mathbf{y}$, when executed on the actual game, or geographical database, produces the desired outcome.

3 Updating the semantic interpreter

In this paper, we advocate the idea of instructable computing, in which an automated learning system is taught a concept via direct instructions rather than by examples. The role of traditional machine learning is therefore shifted from learning the target concept to learning to interpret natural instructions, which explain the target concept.

The learning problem is defined as finding a good set of parameters for the inference function described in Eq. (1), such that, when applied to natural language instructions, the corresponding output formula results in a correct behavior. Typically, such prediction functions are trained in a supervised setting in which the learner has access to training examples, consisting of the input sentences and their corresponding logical forms; we refer to these annotated structures as gold structures, and denote them by $\{(\mathbf{x}^l, \mathbf{y}^l)\}_{l=1}^N$ (Zettlemoyer and Collins 2005; Wong and Mooney 2007). However, our learning framework does not have access to this type of annotation and relies only on feedback obtained from world interaction—by executing the interpretation.

This setup gives rise to our algorithmic learning approach. We perform the following steps iteratively—(1) generating logical formulas from natural language sentences (e.g., card game rules from natural language instructions), (2) and receiving feedback by executing the resulting output formula and updating the interpretation function parameters accordingly. The difference between our approach and a supervised learning approach can be summarized as follows—in a supervised setting the learner is trained over a fixed set of examples, consisting of input objects and the corresponding output structures $\{(\mathbf{x}^l, \mathbf{y}^l)\}_{l=1}^N$. In our setting, the learner is trained iteratively over a set of training examples, consisting of triplets $(\mathbf{x}, \mathbf{y}, b)$, where b is a binary indication for the correctness of $\mathbf{y}$, the predicted structure generated from the input sentence $\mathbf{x}$. In addition, unlike the supervised setting in which the set of training examples is fixed, in our setting at the end of each iteration the predicted structures and the binary feedback associated with them are added to the set of training examples.

Providing an indication of correctness of the overall predicted structure is typically easy to do. Alas, it is not informative enough to be used directly, as it does not provide direct feedback that indicates whether the components constituting the prediction (the semantic parse output) are correct, as required by standard structured learning algorithms. More formally, learning in these settings can be framed as learning a structured predictor[1] using a binary supervision source. Therefore, the key question is: *how can one use the available binary supervision signal to generate the structured feedback signal required for training?*

In this section we aim to answer this question. Our discussion is driven by the observation that the key difficulty in bridging the gap between the binary supervision available and the structured pairs required for learning, stems from the *fault assignment problem*. Supervised learning is guided by a basic principle—minimizing a loss function, defined over the annotated data. In structure learning, the loss function quantifies the incorrect structural decision, penalizing (or assigning fault) only to substructures predicted incorrectly. However in our settings, the learner only has a binary indication of the incorrect prediction, rather than the more informative structural decomposition required. A simple solution (taken for example by Clarke et al. (2010), Liang et al. (2011)), focuses on learning only from predictions receiving positive feedback, by extracting structured signal from these predictions and iteratively training the model.

[1]The learning domains discussed in this paper rely on an intermediate layer, $\mathbf{h}$, which is not included in the training set. Our algorithm deals with this variation effortlessly, however when working in a supervised setting a latent variable variation of these algorithms is required. See Yu and Joachims (2009); Chang et al. (2010a) for details.

Unlike these approaches, our algorithm is designed to use both types of feedback: it uses positive feedback as structural supervision, and negative as binary supervision, thus utilizing the supervision signal more effectively. In Sect. 3.3, we present an online learning algorithm combining both binary learning and structured learning principles. In its most basic form, our algorithm utilizes the negative feedback signal in a coarse way, by penalizing the entire structure (i.e., all the structural decisions) uniformly. In Sect. 3.4 we take an additional step, and show how to amplify the binary signal by approximating the mistakes in the predicted structures that receive negative feedback, thus providing finer grained feedback to the algorithm. We follow the observation that the predicted structure can be naturally decomposed to multiple components, some of which can be predicted more reliably than others and propose an algorithm that better exploits this structural decomposition; specifically, we suggest to approximate the loss (i.e., the incorrect decisions) associated with structures assigned negative feedback by decomposing the structure into individual components and assessing their correctness probability. We can estimate the correctness probability reliably by using the set of positive predictions to compute their statistics.

3.1 Learning structures from binary feedback

In this section we discuss how to learn structures from binary signals in more details. We begin by providing an overview of existing approaches to this problem. These typically rely only on positive feedback by framing the problem as an incremental supervised learning problem. We then proceed to describe our algorithm, and show how to use both structures assigned positive and negative labels. Finally we show how the negative binary feedback can be better exploited to provide finer-grained feedback to the learner instead of uniformly demoting the negative structure.

3.2 Existing approaches for learning with binary feedback

In Algorithm 1 we describe a high level procedure for these settings. The algorithm incrementally samples the space of possible structures for a given input, modifying the model's parameter vector to encourage correct structures and reject incorrect ones. The algorithm repeatedly performs two steps:

Algorithm 1 Existing Approaches: High level view

```
Input: Inputs $\{\mathbf{x}^l\}_{l=1}^N$,
    $Feedback : \mathcal{X} \times \mathcal{Y} \to \{+1, 1\}$,
    initial weight vector $\mathbf{w}$
    {Notation: $B$ set of collected training examples, $(\mathcal{X}, \mathcal{Y})$ space of (inputs,outputs),
    $\phi$ feature function, $f$ boolean variable, $N$ number of examples}
1: repeat
2:   for $l = 1, \dots, N$ do
3:     $\hat{\mathbf{h}}, \hat{\mathbf{y}} = \arg\max_{\mathbf{h},\mathbf{y}} \mathbf{w}^T \Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y})$
4:     $f = Feedback(\mathbf{x}^l, \hat{\mathbf{y}})$
5:     $B = B \cup \{(\Phi(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}), f)\}$
6:   end for
7:   $\mathbf{w} \leftarrow TRAIN(B)$
8: until Convergence
9: return w
```

1. Predict output structures using current model and receive feedback for them (lines 1–6).
2. Train the model on new data generated using this feedback (line 7).

Given the training set collected at the first step, existing algorithms perform the second step by doing one of the following:

Structured update Since positive feedback correspond to correct structural decision, these can be used directly for training a structured supervised model. This instantiation of the algorithm is in essence an incremental supervised learning algorithm, where at each stage more labeled examples are added to the training set. The choice of structured learning procedure used is left to the system designer. For example, previous works used structured SVM and the EM algorithm.

While this procedure is very intuitive, it can only utilize structures assigned positive feedback and it therefore ignores the negative feedback.

Binary update Cast the learning problem as a binary classification problem, where the feature decomposition of correct structures is treated as positive examples and incorrect structures as negative. The weight vector is updated using a binary learning algorithm. This approach allows using both types of feedback, but this feedback is very coarse, it does not take advantage of structural information available for structures assigned a positive label, and could thus suffer from a sparsity problem.

3.3 A combined feedback perceptron

Our algorithm is designed to make use of both the negative and positive feedback, by combining ideas from both structure and binary learning. Unlike supervised learning algorithms, which take a fixed set of training examples, the algorithm iteratively generates its own training data,[2] using its current set of parameters to generate structures, and the feedback function to label them. The algorithm can be considered as a fusion of the binary perceptron (Rosenblatt 1958) and structured perceptron (Collins 2002) and works in an online fashion, performing error driven updates.

Algorithm 2[3] describes this procedure. The algorithm iteratively receives examples x and generates the best corresponding structures given its current set of parameters $\mathbf{w}$ (line 4) and it then receives feedback for the chosen structure (line 5), if the feedback is negative, it incurs a penalty–if the learner, in any of the previous iterations, has generated a positive structure for that input object, the parameter set is updated towards that structure using the structured perceptron update rule (line 8), and if not, the parameter set is updated away from the selected point by using the binary perceptron update rule (line 10).[4]

The algorithm performs two types of updates: a *structural update* is done when a positive structure for that example has already been encountered by the algorithm (we denoted this structure as $\mathbf{h}^*, \mathbf{y}^*$). In this case the algorithm simply performs the structured perceptron updated rule, given as

[2]Since at different iterations, the inference process might return a different output, we only store the latest output, although other caching policies could be used.

[3]Throughout the paper we use the abbreviation *CombPercept* to refer to this algorithm.

[4]The number of negative structures is likely to overwhelm the learning process; to prevent that we restrict the number of binary updates to the size of the positive training set.

Algorithm 2 Combined Feedback Perceptron

```
Input: Sentences $\{\mathbf{x}^l\}_{l=1}^N$,
    $Feedback : \mathcal{X} \times \mathcal{Y} \rightarrow \{+1, 1\}$,
    initial weight vector $\mathbf{w}$
    {Notation: $B_l$ set of collected training examples, $(\mathcal{X}, \mathcal{Y})$ space of (inputs,outputs),
    $\phi$ feature function, $f$ boolean variable, $N$ number of examples}
1:  $B_l \leftarrow \{\}$ for all $l = 1, \ldots, N$
2:  repeat
3:    for $l = 1, \ldots, N$ do
4:      $\hat{\mathbf{h}}, \hat{\mathbf{y}} = \arg\max_{\mathbf{h},\mathbf{y}} \mathbf{w}^T \Phi(\mathbf{x}^l, \mathbf{h}, \mathbf{y})$
5:      $f = Feedback(\mathbf{x}^l, \hat{\mathbf{y}})$
6:      if $f = -1$ then
7:        if $B_l$ contains an entry for $\mathbf{x}^l$ then
8:          $\mathbf{w} \leftarrow$ Update using Structure($\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}$)
9:        else
10:         $\mathbf{w} \leftarrow$ Binary Update($\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}$)
11:       end if
12:     else
13:       add $(\Phi(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}))$ to $B_l$
14:     end if
15:   end for
16: until Convergence
17: return $\mathbf{w}$
```

Structural Update

$$\mathbf{w} = \mathbf{w} + \Phi\left(\mathbf{x}^l, \mathbf{h}^*, \mathbf{y}^*\right) - \Phi\left(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}\right) \tag{2}$$

Alternatively, in case the algorithm does not have a positive structure to update towards, the algorithm demotes all active features uniformly, as described by the following equation.
Simple Binary Update

$$\mathbf{w} = \mathbf{w} - \Phi\left(\mathbf{x}^l, \hat{\mathbf{h}}, \hat{\mathbf{y}}\right) \tag{3}$$

Note that the binary update rule treats the entire structure as incorrect. The rationale behind it is that there is not gold structure to update towards, the "blame" for the negative structure is assigned uniformly. In the following section we describe how to refine this information by approximating which substructures are actually responsible for the negative signal.

3.4 Learning to approximate structural loss

The binary update rule targets a simple objective–updating the current set of parameters, such that the incorrect structure will no longer be ranked highest by the model. This is achieved by uniformly demoting the weights of all features corresponding to active decisions in the incorrect structure. This contrasts with the structured update rule, which aims to "correct" the decision, rather than just change it, by demoting the weights of features corresponding to incorrect decisions, and promoting the weights of those corresponding to correct ones. In the supervised setting, these updates are possible since the gold structure provides this information.

While this information is not available, we try to approximate it by learning a structural loss function based on the set of correct predictions. Our loss function has a probabilistic interpretation, mapping each structural decision in to a probability score. Given an incorrect structure, we use the approximated loss function and update the model according to the score it assigns to structural decisions. Decisions assigned a low probability score are more likely to correspond to the mistakes that have led to the incorrect prediction.

The approximation is done by following the observation that the predicted structure is composed of several components, some more likely to be correct than others based on their occurrence in the positive data. Note that although these components have appeared in correct predictions, it does not necessarily imply that given a new input example, consisting of similar constructs, the same prediction should be repeated, due to syntactic and lexical variance. Some measure of confidence in these components is therefore required.

More formally, given an incorrect structural decision, we decompose the structure into a set of substructures, and evaluate the conditional probability of each structural decision ($y_i \in \mathbf{y}$) given the input generating it. The way in which the output structure decomposes into individual decisions and how input fragments are assigned to them is determined by the feature functions, and by whether parts of the inputs are relevant for a given decision. We use the computed probabilities to appropriately penalize the model: if the decision is likely to be correct (i.e., it is assigned a high probability) the demotion step will have little effect on the weights of features corresponding to that decision and the vice versa.

Accommodating this procedure requires two changes to Algorithm 2. The first change is to the Binary update rule described in line 10 of Algorithm 2.[5] The uniform penalty is replaced by a procedure described in Algorithm 3, which assigns a different penalty to each structural decision based on its confidence score. The second change concerns the computation of the confidence score. This score is computed over the set of examples assigned a positive feedback signal. We change line 13 in Algorithm 2, to consider newly added positive structures in the loss approximation computation. Our algorithm maintains a probability distribution over the individual decisions that appear in the positively predicted structures. Given a new positive structure we decompose it into the individual output decisions and the relevant part of the inputs, and the probability distribution over the two is updated.

In order to understand the intuition behind this approach consider the following (sentence,interpretation) pair.

Example 2 Input:
“What is the population of the largest city in New York?”
Output:
`Population( Largest (City (`**`NEW_YORK_CITY`**` ))).`

In this example the output structure will receive a negative feedback since its interpretation of the term “New York” refers to New York city, rather than the state of New York. By focusing the update step on that decision we can encourage the model to try a different interpretation of that specific term, rather than penalizing potentially correct substructures in the output formula. Continuing with our example, consider two components in the output structure, an incorrect component—the mapping between “New York” and `NEW_YORK_CITY`, and a correct one, mapping between “largest” and the predicate `Largest`. Since “New

[5] Throughout the paper we use the abbreviation *CombPercept w AprxLoss* to refer to the combined perceptron algorithm using this update rule.

Algorithm 3 Approximated Structural Update (Line 10 in Algorithm 2)

```
Input: Input $(\mathbf{x}, \mathbf{h}, \mathbf{y})$
1: for $(x, h, y)_i \in (\mathbf{x}, \mathbf{h}, \mathbf{y})$ do
2:     $\mathbf{w} = \mathbf{w} - \phi((x, h, y)_i) \cdot (1 - p(y_i|x_i))$
3: end for
```

York” is an ambiguous term which could refer to either a state or a city, the probability mass is likely to be divided between all the senses of the term, when computed over the positive training data, and therefore the probability score $p($ **NEW_YORK_CITY** | “New York”) will reflect it, leading to a heavier penalty assigned to this term. Since the term “largest” is typically used to refer to a the same predicate symbol, a lower penalty will be assigned to it.

We observe that the success of our loss approximation method is dependent on the specific structural decomposition for the domain. In domains where the structural decomposition considers *local* feature functions, computing the conditional probability is likely to provide useful information, as these local input-output substructures are likely to appear frequently in the data (as in the example above, where the lexical mapping decision depended only on the term “New York”, which is likely to appear in other examples). An extreme opposite scenario could occur when each substructure relies on the entire input. Fortunately, linguistic input decomposes naturally in to smaller units over which the structural decision is defined, thus allowing our loss approximation approach to gather and provide useful information during learning.

The key idea behind this approach is to bias the model towards repeating sub-structures, under the assumption that repeating patterns tend to be consistent when sampled from the same domain. This results in biasing the update procedure to allow the model to focus on new, or ambiguous, patterns. It should be noted that this method does not guarantee an improved update procedure, since it can reduce to a uniform penalty, and could even penalize correct structures more heavily than incorrect ones. However in our empirical evaluation, it was shown to be useful (see Sect. 5 for experimental results and analysis).

4 Semantic interpretation

Semantic parsing is the process of converting a natural language input into a formal logic representation. This process is performed by associating lexical items and syntactic patterns with logical fragments and composing them into a complete formula. Existing approaches rely on extracting, from annotated training examples, a set of *parsing rules*, that map text constituents to a logical representation and applying them recursively to obtain the meaning representation. Adapting to new data is a major limitation of these approaches as they cannot handle inputs containing syntactic patterns which were not observed in the training data.

For example, assume the training data produced the following set of parsing rules for the Geoquery domain. Each rule consists of a right hand and left hand side. Each consists of either a lexical element (e.g., “capital”), or a combination of a syntactic category and a logical formula (e.g., NP *[*λ`x.capital(x)`*]*).

Example 3 Typical parsing rules
(1) NP *[*λ`x.capital(x)`*]* $\rightarrow$ capital
(2) PP *[* `const(texas)`*]* $\rightarrow$ of Texas
(3) NNP *[* `const(texas)`*]* $\rightarrow$ Texas

(4) NP *[*`capital(const(texas))`*]* →
NP*[*λ`x.capital(x)`*]* PP *[* `const(texas)`*]*

Rules 1–3 describe lexical transformations, these rules are triggered by a raw input fragment (such as a word, or a short phrase) and generate a logical symbol and a syntactic category (such a Noun Phrase, Preposition Phrase etc.). Rule 4 describes a higher level transformation, in which two pairs of syntactic category and matching logical fragments are unified into a single syntactic category and logical formula.

Given a sentence (such as the ones in Example 4) the meaning of a sentence is constructed by applying these rules recursively. It can be observed that despite the lexical similarity in these examples, the semantic parser will correctly parse the first sentence but fail to parse the second because the lexical items belong to a different syntactic category (i.e., the word *Texas* is not part of a preposition phrase in the second sentence). The third sentence will fail to parse due to missing lexical information—the term "Longhorn State" is not covered by any of the rules.

Example 4 Syntactic variations of the same meaning representation
Target logical form: `capital(const(texas))`
Sentence 1: "What is the capital of Texas?"
Sentence 2: "What is Texas' capital?"
Sentence 3: "What is the capital of the Longhorn State?"

The ability to adapt to unseen inputs is one of the key challenges in semantic parsing. Several works (Zettlemoyer and Collins 2007; Kate 2008) have suggested partial solutions to this problem, for example by manually defining syntactic transformation rules that can help the learned parser generalize better.

Given the previous example (sentence 2), we observe that it is enough to identify that the function `capital(·)` and the constant `const(texas)` appear in the target logical interpretation, since there is a single way to compose these entities into a single formula—`capital(const(texas))`.

Motivated by this observation we define our meaning derivation process over the rules of the domain interpretation language and use syntactic information as a way to bias the logical interpretation process. That is, our inference process considers the *entire* space of meaning representations irrespective of the patterns observed in the training data. This is possible as the logical interpretation languages are defined by a formal language and formal grammar.[6] The syntactic information present in the natural language is used as soft evidence (features) which guides the inference process to good meaning representations.

In addition, we use existing external knowledge resources capturing lexical information to make up for missing lexical information. In this case (sentence 3), mapping between "Texas" and "Longhorn State".

This formulation is a major shift from existing approaches that rely on extracting parsing rules from the training data. In existing approaches the space of possible meaning representations is constrained by the patterns in the training data and syntactic structure of the natural language input. Our formulation considers the entire space of meaning representations and allows the model to adapt to previously unseen data; this way we always produce a semantic interpretation, and the one produced is biased to cohere with patterns observed in the input.

[6]This is true for all meaning representations designed to be executed by a computer system.

We frame our semantic interpretation process as a constrained optimization process, maximizing the objective function defined by Eq. (1). The main part of this objective function is the feature mapping ϕ that relies on extracting lexical and syntactic features instead of parsing rules. In the remainder of this section we explain the components of our inference model.

4.1 Semantic interpretation as constrained optimization

Semantic interpretation, as formulated in Eq. (1), is an inference procedure that selects the top ranking output logical formula. In practice, this decision is decomposed into smaller decisions, capturing local mappings of input tokens to logical fragments and their composition into larger fragments. These decisions are converted into a feature representation by a feature function Φ, and is parameterized by a weight vector.

We formulate the inference process over these decision as an Integer Linear Program (ILP), maximizing the overall score of active decisions, subject to constraints ensuring the validity of the output formula. The flexibility of ILP has previously been advantageous in natural language processing tasks (Roth and Yih 2004, 2007; Martins et al. 2009) as it allows us to easily incorporate constraints declaratively. These constraints help facilitate learning as they shape the space of possible output structures, thus requiring the learned model's parameters to discriminate between a smaller set of candidates. The flexibility offered by using an ILP solver comes with a computational cost—ILP is in general, NP-hard. In this work we used an off-the-shelf solver[7] incurring the full computational cost, which while efficient for the most part, restricted us in some of the experiments we performed. We consider using approximation techniques and ILP-relaxation techniques as future work.

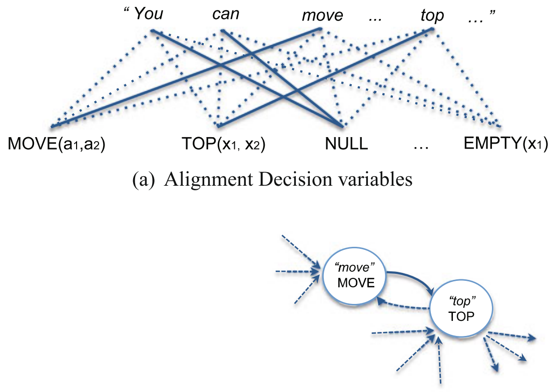
4.2 Decision variables and objective function

The inference decision is defined over two types of decision variables. The first type, referred to as *alignment decision variables* (abbreviated as *a-var*), encodes a lexical mapping decision as a binary variable α_{cs}, indicating that a constituent c is aligned with a logical symbol $\mathtt{s}$. The pairs connected by the alignment ($\mathbf{h}$) in Fig. 3(a) are examples of such decisions.

The final output structure $\mathbf{y}$ is constructed by composing individual predicates into a complete formula. This is formulated as an *argument sharing decision* indicating if two functions take the same variable as input. We refer to this type of decisions as *composition decision variables* (abbreviated as *c-var*), encoded as a binary variable, β_{cs^i,dt^j} indicating if the j-th argument of $\mathtt{t}$ (associated with constituent d) and the i-th argument of $\mathtt{s}$ (associated with constituent c) refer to the same variable or constant symbol. For example, the decision variables representation of the formula presented in Fig. 3(b): $\mathtt{move}(a_1, a_2)$ $\mathtt{top}(a_1, x_2)$ includes an active c-var indicating that the corresponding predicates share an argument.

Objective function Given an input sentence, we consider the space of possible semantic interpretations as the space of possible assignments to the decision variables. The semantic interpretation decision is done by selecting a subset of variables maximizing, subject to constraints, a linear objective function, defined as follows

[7] We used the ILP solver Xpress-MP in our experiments.



(a) Alignment Decision variables

(b) Compositional Decision variables

(c) Flow variables

Fig. 3 An example of inference variables space for a given input. The dashed edges correspond to non-active decision variables and the bold lines to active variables, corresponding to the output structure $\texttt{move}(a_1, a_2)$ $\texttt{top}(a_1, x_2)$. Active variables include—a-vars: $\alpha_{(\text{“}move\text{”},\texttt{move}^1)}$, $\alpha_{(\text{“}top\text{”},\texttt{top}^1)}$, c-vars: $\beta_{(\text{“}move\text{”},\texttt{move}^1),(\text{“}top\text{”},\texttt{top}^1)}$, and positive flow: $f_{(\text{“}move\text{”},\texttt{move}^1),(\text{“}top\text{”},\texttt{top}^1)}$

$$
\begin{aligned}
F_{\mathbf{w}}(\mathbf{x}) = \arg\max_{\alpha,\beta} & \sum_{c\in\mathbf{x}} \sum_{s\in D} \alpha_{cs} \cdot \mathbf{w_1}^T \Phi_1(\mathbf{x}, c, s) \\
& + \sum_{c,d\in\mathbf{x}} \sum_{s,t\in D} \sum_{i,j} \beta_{cs^i,dt^j} \cdot \mathbf{w_2}^T \Phi_2\big(\mathbf{x}, c, s^i, d, t^j\big)
\end{aligned}
\tag{4}
$$

where i, j iterate over `args(s)` and `args(t)` respectively, and D is the set of logical symbols in the domain.

4.3 Constraints

Given an input sentence, the space of possible interpretations is determined by the space of possible assignments to the decisions variables. However, there is a clear dependency between α-variables and β-variables assignments, as functions can only share a variable (β decision) if they appear in the output formula (α decisions), for example, and these dependencies restrict the space of feasible assignments.We take advantage of the flexible ILP framework, and encode these restrictions as global constraints over Eq. (4). Next we describe the constraints used in the formulation.

Lexical mapping decisions

– An input constituent can only be associated with at most one logical symbol.

$$\forall c \in \mathbf{x}, \quad \sum_{s\in\mathbf{D}} \alpha_{cs} \le 1$$

– The head predicate (e.g., `move`) must be active.

$$\sum_{c\in\mathbf{x}} \alpha_{c,head} = 1$$

Argument sharing decisions

– Variable sharing is only possible when variable types match.
– If two predicates share a variable, then these predicates must be active.

$$\forall c, d \in \mathbf{x}, \forall s, t \in \mathbf{D}, \quad \beta_{cs^i, dt^j} \implies \alpha_{cs} \wedge \alpha_{dt}$$

where i, j range over `args(s)` and `args(t)` respectively.

Global connectivity constraints In addition to constraints over local decisions we are also interested in ensuring that the output formula has a correct global structure. We impose constraints forcing an overall *fully-connected* output structure, in which each logical symbol appearing in the output formula is connected to the head predicate via argument sharing. This property ensures that the value of each logical construct in the output is dependent on the head predicate's arguments. In order to clarify this idea, consider the output logical formula described in Example 5. We consider it to be an illegitimate formula. The value of that formula, when given a game state, is evaluated to `true` if the game state contains at least one vacant freecell, *not necessarily the target freecell* specified by the head predicate arguments.

Example 5 (Disconnected Output Structure)
$\texttt{move}(a_1, a_2) \leftarrow \texttt{top}(a_1, x_1) \wedge \texttt{card}(a_1) \wedge \texttt{freecell}(x_2) \wedge \texttt{empty}(x_2)$

Note that this constraint does not limit the model's effective expressivity as it only rules out interpretations that are illegitimate or as in Example 5, where the target of the move may not be an empty free cell. From a natural language processing point of view, this process is very similar to the co-reference resolution problem of ensuring that all mentions refer back to an originating entity.

We encode the connectivity property by representing the decision space as a graph, and forcing the graph corresponding to the output prediction to have a connected tree structure using flow constraints. Let $G = (V, E)$ be a directed graph, where V contains vertices corresponding to α variables and E contains edges corresponding to β variables, each adding two directional edges. We refer to vertices corresponding to $\alpha_{c,\texttt{move}}$ variables as head vertices. Clearly, the output formula will be fully-connected if and only if the graph corresponding to the output structure is connected. We associate a flow variable f_{cs^i, dt^j} with every edge in the graph, and encode constraints over the flow variables to ensure that the resulting graph is connected.

Figure 3(c) provides an example of this formulation: the two nodes represent alignment decisions, connecting lexical items and logical symbols. These two nodes are connected by two edges, representing the compositional decision. We associate an additional flow variable with the edges. Note that the flow variables take integer values, unlike other ILP variables discussed so far that represent boolean decisions. In order to ensure a connected structure we restrict the values of flow variables, in this simple example, if we assume that both alignment decision variables are active (i.e., they represent active mappings to logical symbols), all that is left to ensure connectivity is to force the flow to be equal to 1, or more broadly—the number of active nodes in the graph (not including the head node). We can encode these restrictions easily as ILP constraints.

– Only active edges can have a positive flow.

$$\beta_{cs^i, dt^j} = 0 \implies f_{cs^i, dt^j} = 0 \wedge f_{dt^j, cs^i} = 0$$

– The total outgoing flow from all head vertices must be equal to the number of logical symbols appearing in the formula.

$$\sum f_{*,\mathtt{move}^i,*,*} = \sum_{c \in \mathbf{x}} \sum_{s \in \mathbf{D} \setminus \{\mathtt{move}\}} \alpha_{cs}$$

For readability reasons, we use * to indicate all possible values for constituents in **x** and logical symbols in **D**.

– Each non-head vertex consumes one unit of flow.

$$\forall d \in \mathbf{x}, \forall t \in \mathbf{D}, \quad \sum f_{*,*,dt^j} - \sum f_{dt^j,*,*} = 1$$

4.4 Features

The inference problem defined in Eq. (4) uses two feature functions: Φ_1 for alignment decision variables and Φ_2 for compositional decisions variables. In general, Φ_1 represents lexical information while Φ_2 represents syntactic and semantic dependencies between substructures.

Alignment decision features Φ_1 Determining if a logical symbol is aligned with a specific constituent depends mostly on lexical information. Following previous work (e.g., Zettlemoyer and Collins 2005) we create a small lexicon, mapping logical symbols to surface forms. We initialize the lexicon by mapping the logical entities to lexical items carrying the same name (e.g., `card`(·) was mapped to "card"). We extend the lexicon during the learning process–whenever the model predicts a logical formula assigned a positive feedback, the lexical choices done by the model are added to the lexicon. For example, given an alignment **h** that maps the word "neighboring" to the predicate `border`(·), the pair will be added to the lexicon.

We rely on an external knowledge base, WordNet (Miller et al. 1990), to extend the initial lexicon and add features which measure the lexical similarity between a constituent and a logical symbol's surface forms (as defined by the lexicon). In order to disambiguate preposition constituents, an additional feature is added. This feature considers the current lexical context (one word to the left and right) in addition to word similarity.

Compositional decision features Φ_2 Compositional decisions rely on syntactic information. We use the dependency tree (Klein and Manning 2003) of the input sentence. Given a compositional decision $\beta_{cs,dt}$, the dependency feature takes the normalized distance between the head words in the constituents c and d.

In addition, a set of features indicate which logical symbols are usually composed together, without considering their alignment to text. These features allow the model to recover in cases where the input sentence contains unknown lexical items. Since no lexical evidence is available, these features allow the model to take an informed guess based on previous decisions. For example, given the input sentence "*what is the population of NY?*", the interpretation of the ambiguous term "NY" as a city would result in a compositional decision `population`(`city`(·)), and its interpretation as a state would result with the decision `population`(`state`(·)). Each of the competing compositional decisions would produce a feature capturing the logical output of each decision, which would bias the decision towards the output structure that appears more frequently in the training data.

5 Experiments

In this section we describe our experimental evaluation.[8] We begin this section by describing our experiments on instructional text for several variations of the Solitaire card game. In this domain we evaluated the results of applying our overall approach, teaching an agent the rules to a card game, treating the predicted structure as a classification rule and evaluating the result on card game moves.

In the second part of this section we focus on response based learning, and evaluate our approach on the well known Geoquery domain. In the Geoquery domain the predicted structure is a database query. This well studied domain allows us to compare our results to existing work and evaluate different properties of response based learning.

5.1 Overall approach: teaching solitaire card game rules

Experimental setup The decision function described by the text classifies the legality of several Solitaire card game moves given a game state. The input sentences were taken from solitaire game instructions appearing on-line, with some modifications to accommodate the learning scenario (e.g., breaking instructions down to individual rules and removing irrelevant parts of the instructions).

We consider several games, each having one or more such rules. All the rules describe a similar operation—moving a card from one location to another. The rules differ according to the game and the source and target cards locations. Consider for example two such rules for the famous Freecell solitaire game—FREECELL (move a card to a freecell) and TABLEAU (move a card to a tableau).

Example 6 (FREECELL concept and its description)
$\mathtt{move}(a_1, a_2) \leftarrow \mathtt{top}(a_1, x_1) \wedge \mathtt{card}(a_1) \wedge \mathtt{freecell}(a_2) \wedge \mathtt{empty}(a_2)$

- “*You can move any of the top cards to an empty freecell*”
- “*Any playable card can be moved to a freecell if it is empty*”

Example 7 (TABLEAU concept and its description)
$\mathtt{move}(a_1, a_2) \leftarrow \mathtt{top}(a_1, x_1) \wedge \mathtt{card}(a_1) \wedge \mathtt{tableau}(a_2) \wedge \mathtt{top}(x_2,a_2) \wedge \mathtt{color}(a_1, x_3) \wedge \mathtt{color}(x_2, x_4) \wedge \mathtt{not\text{-}equal}(x_3, x_4) \wedge \mathtt{value}(a_1, x_5) \wedge \mathtt{value}(x_2, x_6) \wedge \mathtt{successor}(x_5, x_6)$

- “*A top card can be moved to a tableau if it has a different color than the color of the top tableau card, and the cards have successive values*”

It can be easily observed that the rule in Example 6 is considerably easier to predict than the one described in Example 7. Both the input sentence and the predicted output structure are simpler. We can therefore expect a variability in the results based on the complexity of the rules.

In order to evaluate our framework we associate with each target concept instructional text describing the target rule, along with game data over which the predicted structures are evaluated. Each target concept (e.g., move a card to a free cell) is associated with 10 different textual instructions describing it, with variations in both sentence structure and

[8] The data used in this paper will be made available at: http://cogcomp.cs.illinois.edu/page/resources_exp2.

lexical choices. In addition, in order to provide feedback to the learning process we provided a set of 10 labeled *game* examples for each rule, 5 positive and 5 negative. Testing the correct interpretation of the rule was done by sampling 900 game moves. Since the space of possible moves consists of mostly illegal moves, we biased the sampling process to ensure that the test set contains an equal number of positive and negative examples. Note that these 900 examples were used for evaluation only.

We evaluated the performance of our learning system by measuring the proportion of correct predictions for each of the target concepts on the game data. The accuracy for each target concept is measured by averaging the accuracy score of each of the individual instruction interpretations.

The semantic interpreter was initialized using a simple rule based procedure, assigning uniform scores to input constituents appearing in the lexicon (a-vars) and penalizing compositional decisions (c-vars) corresponding to input constituents which are far apart on the dependency tree of the input sentence.

Experimental approach Our experiments were designed to evaluate the learner's ability to generalize beyond the limited supervision offered by the feedback function. The term generalization can be interpreted in two different ways, it can refer to the quality of the learned *target concept* described by the natural language instructions (e.g., move a card to a free cell) or to the quality of the learned semantic parser. We design two experiments to evaluate our learning approach in light of the two notions of generalization. In the first, we assume the learner has supervision, via the binary feedback function, for the target concept described by the natural language instruction. In the second experiment, this feedback is not available, and the system relies on a semantic parser that was learned, using the binary feedback function, while interpreting other tasks.

(1) Evaluating the quality of the learned target concept: the ability of the system to correctly classify previously unseen solitaire game moves. The game-related resources used in the training process amount to the small and fixed set of labeled game moves used by the feedback function. A very intuitive notion of over-fitting the model to the training data is constructing an output formula that can only classify the game moves observed during training. Testing the resulting output formula on previously unseen game moves evaluates this notion of generalization directly. In some sense this is a measure of how well the natural language lesson succeeded in biasing the learning process away from the direct feedback provided by the *grounded* labeled examples used by the feedback function. The learning process in this scenario is applied to each rule using its feedback function. Since rules use a common lexicon we learned these rules together (i.e., updating a shared weight vector when learning all the rules sequentially).
(2) Evaluating the quality of the learned semantic interpretation model. Our goal goes beyond learning the rules for a single task: the semantic interpreter is meant to provide a broader interpretation ability, to the extent it can be applied, after learning one task, to other tasks defined over a similar lexicon. To study this scenario, after the learning process terminated, the system was given a set of new textual instructions describing a previously unseen game rule, and its performance was evaluated based on the quality of the newly generated rules. To accommodate this setup we performed a 10 fold cross-validation over the data and report the averaged results over the target rule's interpretations. In this case cross-validation refers to leaving one rule out during the learning process, and then at test time, averaging the results over all the textual variations of that rule.

Table 2 Testing generalization over Solitaire *game data*. Each rule is described by the game it belongs to (larger font), and the specific rule name in that game (smaller font). Accuracy was evaluated over previously unseen game moves using the classification rules learned from the instructions used in training. The Initial Model column describes the performance of the rules generated by the initial interpretation model (i.e., before learning)

Target concept	Initial model	Learned model
FREECELL freecell	0.76	0.948
FREECELL Homecell	0.532	0.686
FREECELL Tableau	0.536	0.641
ACCORDION consecutive left	0.64	0.831
ACCORDION three cards gap	0.561	0.724
AGREEMENT	0.74	0.932
ALHAMBRA King	0.61	0.786
ALHAMBRA Ace	0.632	0.764
ACES UP Between	0.76	0.924
ACES UP Out	0.591	0.691

Table 3 Testing generalization over Solitaire game *textual rules*. Each rule is described by the game it belongs to (larger font), and the specific rule name in that game (smaller font). Accuracy was evaluated over previously unseen game moves using classification rules generated from *previously unseen game instructions*. Semantic interpretation was done using the learned semantic interpreter

Target concept	Initial model	Learned model
FREECELL freecell	0.76	0.948
FREECELL Homecell	0.532	0.679
FREECELL Tableau	0.536	0.635
ACCORDION consecutive left	0.64	0.817
ACCORDION three cards gap	0.561	0.678
AGREEMENT	0.74	0.932
ALHAMBRA King	0.61	0.763
ALHAMBRA Ace	0.632	0.753
ACES UP Between	0.76	0.924
ACES UP Out	0.591	0.631

Results Our results are summarized in Tables 2 and 3, the first describing the ability of our learning algorithm to generalize to new *game data*, and the second, to *new instructions*.

A natural baseline for the prediction problem is to simply return `false` (or `true`) regardless of the input—this ensures a baseline performance of 0.5. The system's performance when using only the initialized model without learning improves over the simplified baseline, considerably in some cases, and only barely in others. After learning results consistently improve for all rules.

As can be noticed in examples 6 and 7 described above, target concepts have different levels of difficulty. For example, the FREECELL concept is relatively easy compared to the other rules, both in terms of the output structure and the text used to describe it. The results indeed support this observation, and performance for this task is excellent. The other tasks are more difficult, resulting in a more modest improvement; however, the improvement due to learning is still clear.

In the second scenario we tested the ability of our semantic interpreter to generalize to previously unseen tasks. In this case successful learning depends on the different learning tasks sharing similar constructs and lexical items. We can expect performance to drop in this case, even simply for the reason that the semantic interpreter is trained using less data.

Nonetheless, the question remains—does the improvement obtained after learning stems only from the guidance provided by the concrete game data examples provided (via the feedback function), or does LNI actually supported the development of a semantic parser in the course of learning the rules of the game? If the improvement is only due to supervision offered by the labeled game moves used by the feedback function, we can expect performance to drop significantly as in these new settings the system is evaluated on a classification rule it generated after training, without feedback from the domain while constructing it.

Table 3 shows the performance achieved in this challenging learning scenario. It can be observed that while there is some performance drop, generally our learning algorithm is also able to learn a good semantic interpreter, which generalizes well to previously unseen instructions. Interestingly, the difference in performance compared to the first setting (Table 2) differs depending on which rule is tested. Simple rules achieve good score in both cases, and their performance does not change. We hypothesize that this is due to the fact that the text corresponding to these rules is relatively simple and the model can predict the correct rule even without game data. As rules get harder, a specialized learning process is required. This can be observed by the drop in performance. Most notably this happens in the case of ACCORDION THREE CARDS GAP which uses a predicate not used in any of the other rules.

5.2 Understanding learning with binary supervision: geoquery domain

In order to get a better understanding of our response based learning procedure we used the Geoquery dataset, and compared the performance of several variations of our algorithm, trained over datasets of varying sizes. The dataset we used contains 250 queries used for training, and additional 250 used for testing.[9]

5.3 Empirical results

We compare the results obtained by our algorithm to two natural reference points. The first is the initial model used to bootstrap learning (denoted INITIAL MODEL), improving significantly over this baseline is required to demonstrate effective learning. The second reference point is a supervised model, trained over the same inputs, however using annotated structures rather than binary feedback. We measure performance using the Accuracy score. We refer to our combined learning algorithm as COMBPERCEPT, and when applied with loss approximation we use the term COMBPERCEPT W APRXLOSS.

The results for the Semantic Interpretation domain are summarized in Table 4. The initial model used for bootstrapping (INITIAL MODEL), resulted in an accuracy of 22.2 %. Using our algorithm, without using loss-approximation, resulted in an accuracy score of 79.6 %, and with loss-approximation, performance was improved to—81.6 %.

This model outperforms other systems working under this learning scenario that use the same dataset. We first consider the two models used in Clarke et al. (2010). The first uses both types of feedback, using binary updates only (see Sect. 3 for details) and achieves a score of 69.2 %; the second uses structural updates but utilizes only the positive feedback and achieves a score of 73.2 %. The model presented by Liang et al. (2011) combines an effective semantic inference procedure with a learning procedure but utilizes only structures receiving positive feedback. Their model resulted in a score of 78.9 % for this dataset, bootstrapped using the similar lexical resources.[10]

[9]This dataset was used by several previous works (Clarke et al. 2010; Goldwasser et al. 2011; Liang et al. 2011). While other datasets exist for Geoquery, most notably Geo880, this smaller dataset allowed us to learn efficiently and try out many variations of our algorithm, while providing external reference points to compare our results to.

[10]Liang et al. (2011) presents an additional model which uses a larger manually constructed lexicon, resulting in a considerable improvement. Since we did not have the additional information we did not compare to it.

Table 4 Results for the semantic interpretation domain, comparing several models learning with Binary supervision. Our approach outperforms these models when trained over the same datasets and resources. Note that our combined model (*CombPercept*) always outperform competing models using only one type of feedback, and that loss-approximation (*CombPercept w AprxLoss*) improves the results even further

Algorithm	Sup.	Acc.
Initial Model	–	22.2
Binary Supervision (separate)		
Clarke et al. (2010) (Bin)	250 ans.	69.2
Clarke et al. (2010) (St)	250 ans.	73.2
Liang et al. (2011)	250 ans.	78.9
Binary Supervision (combined)		
CombPercept	250 ans.	79.6
CombPercept w AprxLoss	250 ans.	**81.6**
Supervised		
Clarke et al. (2010)	250 strct.	80.4
Wong and Mooney (2006)	310 strct.	60.0
Wong and Mooney (2007)	310 strct.	75.0
Zettlemoyer and Collins (2005)	600 strct.	79.3
Zettlemoyer and Collins (2007)	600 strct.	86.1
Wong and Mooney (2007)	800 strct.	86.6

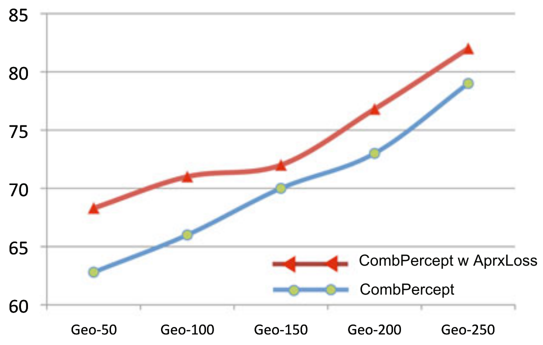


Fig. 4 Comparing the results of the two variations of our combined-perceptron algorithm—with, and without loss-approximation, over datasets of increasing size in the semantic interpretation domain. Results show consistent improvement when using loss-approximation (*CombPercept w AprxLoss*)

There are many works operating in supervised settings. Our model outperforms the supervised model presented in Clarke et al. (2010), and other supervised models (Wong and Mooney 2006, 2007) when trained over dataset of a similar size. This is not surprising, since during training, as the model collects more positive training examples, it converges towards a supervised learning framework.

In order to get a better understanding of the improvement achieved by the loss-approximation framework our algorithm uses, we compared the results of training the two variations of our algorithm (with and without the loss approximation) on datasets of varying sizes. Figure 4 presents the results of these experiments and shows consistent improvement when using loss-approximation.

6 Related work

In this work we study a novel learning protocol based on learning from instructions given by a human teacher. Instructable computing approaches leveraging human expertise are often studied in a reinforcement learning setting, in which a human teacher provides feedback to the learning process (a few recent examples include (Isbell et al. 2006; Knox and Stone 2009; Thomaz and Breazeal 2006)). The role of human intervention in our learning framework is different, as we simulate a natural learning lesson scenario. The approach closest to ours is described in Kuhlmann et al. (2004), integrating the interpretation of natural language advice into a reinforcement learner. However, in their setting the language interpretation model is trained independently from the learner in a fully supervised process.

Converting natural language sentences into a formal meaning representation is referred to as *semantic parsing*. This task has been studied extensively in the natural language processing community, typically by employing supervised machine learning approaches. Early works (Zelle and Mooney 1996; Tang and Mooney 2000) employed inductive logic programming approaches to learn a semantic parser. More recent works apply statistical learning methods to the problem (Kate and Mooney 2006; Wong and Mooney 2007; Zettlemoyer and Collins 2005, 2009; Kwiatkowski et al. 2010). These works rely on annotated training data, consisting of sentences and their corresponding logical forms.

We learn to interpret natural language instructions from feedback given by executing game moves, or by querying a database, instead of supervised learning. Taking this approach helps alleviate the cost of providing the proper supervision for constructing a semantic parser. Learning in similar settings for semantic interpretation has been studied recently by several works: (Chen and Mooney 2008; Liang et al. 2009; Branavan et al. 2009; Tellex et al. 2011; Chen 2012; Kim and Mooney 2012). These works deal with the problem of natural language grounding in a concrete external environment, represented symbolically, and use an external world context as a supervision signal for semantic interpretation. However the semantic interpretation task is different than ours, as the natural language input is completely situated in an external world state. For example, in Chen and Mooney (2008), Liang et al. (2009) the natural language input describes robotic soccer events, such as a player kicking a ball. In this work the natural language input describes a high level rule abstracting over specific states, and is therefore more difficult to interpret.

Several other works learn a semantic parser using other indirect supervision sources. In Matuszek et al. (2012), the authors consider the natural language grounding problem in physical perception, rather then symbolic one. In Vogel and Jurafsky (2010) the authors use a reinforcement learning framework to train an agent to follow navigational instructions. In Artzi and Zettlemoyer (2011) the authors approximate the structural loss using conversational cues. Modeling language acquisition using indirect supervision was studied at Connor et al. (2012) where a semantic-role labeler was learned using partial supervision.

Most relevant to our work are (Clarke et al. 2010; Liang et al. 2011; Goldwasser and Roth 2011) which use these settings for learning a semantic parser. We show how to extend the learning protocol in order to better exploit the binary feedback.

The connection between structured prediction and binary classification over structural decisions was studied in Chang et al. (2010b). In their settings a global optimization objective was defined over a *fixed* set of annotated structures and labeled binary examples. Our algorithm on the other hand does not have any annotated structures, but rather creates its own dataset iteratively by receiving feedback.

Leveraging textual instructions to improve game rules learning was pioneered by Eisenstein et al. (2009) for Freecell Solitaire. In that work textual interpretation was limited to mining repeating patterns and using them as features for learning the game rules over considerable amounts of game training data. Incorporating natural language advice in a game playing framework was also studied (Branavan et al. 2011). In their settings, text interpretation is used to augment the state space representation in a reinforcement learning framework.

7 Conclusions

In this paper we investigate the feasibility of a new type of machine learning based on language interpretation rather than learning only from labeled examples. This process, motivated by human learning processes, takes as input a natural language lesson describing the target concept and outputs a logical formula capturing the learning system understanding of the lesson. This approach has both theoretical and practical advantages, as it reduces the annotation cost and positions the learning process in a way that requires human-level task expertise rather than machine learning and technical expertise.

Learning from Natural Instructions shifts the weight of learning to semantic interpretation and therefore requires a robust semantic interpreter which can be learned without incurring the cost of learning a semantic parser in a supervised fashion. We suggest a training procedure for semantic interpretation that is based on interaction done in the context of the domain of interest. To further facilitate the interpretation process we introduce a lightweight interpretation process that is driven by lexical and syntactic cues, rather than parsing rules.

To fulfill its promise, this type of learning requires communicating effectively with the learning system in a natural, human-level manner. This introduces the major challenge in lesson based learning: interpreting natural language instructions. To avoid the difficulty of training a semantic interpreter independently, we introduce a novel learning algorithm that learns both tasks jointly by exploiting the dependency between the *target concept* learning task and the *language interpretation* learning task.

Following the dual role of learning in LNI, we design our experiments to evaluate the system's ability to *generalize* under both definitions. We begin by showing that our system is able to learn the *target concept* described by the instructions using behavioral feedback, by testing the classification rule resulting from interpreting the natural language instructions, on previously unseen instances. We then proceed to show that the learned semantic parser can be used to interpret previously unseen instructions without any additional supervision directing the new interpretation task. In both cases we show that the system was able to generalize beyond the supervision it received.

In our experiments we exploit the fact that the different concepts, described in natural language, used a similar vocabulary. Although each instruction described a different classification rule, all instructions used terms relevant to card games. Building on that fact the semantic parser trained on one task was able to generalize to previously unseen tasks. In the future we intend to study how to leverage the parser learned for one task, when approaching a new task defined over a different vocabulary and a different set of output symbols. In Goldwasser and Roth (2013) we made a first step in this direction, where we show how to separate the interpretation process into domain-specific and domain-independent components, and reuse the domain-independent information when approaching a new interpretation task.

References

Artzi, Y., & Zettlemoyer, L. (2011). Bootstrapping semantic parsers from conversations. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Branavan, S., Chen, H., Zettlemoyer, L.S., & Barzilay, R. (2009). Reinforcement learning for mapping instructions to actions. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Branavan, S., Silver, D., & Barzilay, R. (2011). Learning to win by reading manuals in a Monte-Carlo framework. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Chang, M., Goldwasser, D., Roth, D., & Srikumar, V. (2010a). Discriminative learning over constrained latent representations. In *Proceedings of the annual meeting of the North American association of computational linguistics (NAACL)*.

Chang, M., Goldwasser, D., Roth, D., & Srikumar, V. (2010b). Structured output learning with indirect supervision. In *Proceedings of the international conference on machine learning (ICML)*.

Chen, D. (2012). Fast online lexicon learning for grounded language acquisition. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Chen, D., & Mooney, R. (2008). Learning to sportscast: a test of grounded language. In *Proceedings of the international conference on machine learning (ICML)*.

Clarke, J., Goldwasser, D., Chang, M., & Roth, D. (2010). Driving semantic parsing from the world's response. In *Proceedings of the conference on computational natural language learning (CoNLL)*.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, *15*, 201–221.

Collins, M. (2002). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Connor, M., Fisher, C., & Roth, D. (2012). *Starting from scratch in semantic role labeling: early indirect supervision*. Berlin: Springer.

Eisenstein, J., Clarke, J., Goldwasser, D., & Roth, D. (2009). Reading to learn: constructing features from semantic abstracts. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Goldwasser, D., & Roth, D. (2011). Learning from natural instructions. In *Proceedings of the international joint conference on artificial intelligence (IJCAI)*.

Goldwasser, D., & Roth, D. (2013). Leveraging domain-independent information in semantic parsing. In *ACL*.

Goldwasser, D., Reichart, R., Clarke, J., & Roth, D. (2011). Confidence driven unsupervised semantic parsing. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Isbell, C., Kearns, M., Singh, S., Shelton, C., Stone, P., & Kormann, D. (2006). Cobot in lambdamoo: an adaptive social statistics agent. In *Proceedings of the international joint conference on autonomous agents and multiagent systems (AAMAS)*.

Kate, R. (2008). Transforming meaning representation grammars to improve semantic parsing. In *Proceedings of the conference on computational natural language learning (CoNLL)*.

Kate, R., & Mooney, R. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Kim, J., & Mooney, R. (2012). Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Klein, D., & Manning, C. (2003). Fast exact inference with a factored model for natural language parsing. In *NIPS*.

Knox, B., & Stone, P. (2009). Interactively shaping agents via human reinforcement. In *KCAP*.

Kuhlmann, G., Stone, P., Mooney, R., & Shavlik, J. (2004). Guiding a reinforcement learner with natural language advice: initial results in robocup soccer. In *AAAI workshops*.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2010). Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Liang, P., Jordan, M., & Klein, D. (2009). Learning semantic correspondences with less supervision. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Liang, P., Jordan, M., & Klein, D. (2011). Learning dependency-based compositional. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Martins, A., Smith, N., & Xing, E. (2009). Concise integer linear programming formulations for dependency parsing. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., & Fox, D. (2012). A joint model of language and perception for grounded attribute learning. In *Proceedings of the international conference on machine learning (ICML)*.

Miller, G., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1990). Wordnet: an on-line lexical database. *International Journal of Lexicography*, *3*, 235–244.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Reviews*, *65*(6), 386–408.

Roth, D., & Yih, W. (2004). A linear programming formulation for global inference in natural language tasks. In H. T. Ng & E. Riloff (Eds.), *Proceedings of the conference on computational natural language learning (CoNLL)*.

Roth, D., & Yih, W. (2007). Global inference for entity and relation identification via a linear programming formulation. In L. Getoor & B. Taskar (Eds.), *Introduction to statistical relational learning*.

Tang, L., & Mooney, R. (2000). Automated construction of database interfaces: integrating statistical and relational learning for semantic parsing. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Tang, L., & Mooney, R. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the European conference on machine learning (ECML)*.

Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., Teller, S., & Roy, N. (2011). Approaching the symbol grounding problem with probabilistic graphical models. *AI Magazine*, *32*(4), 64–76.

Thomaz, A., & Breazeal, C. (2006). Reinforcement learning with human teachers: evidence of feedback and guidance with implications for learning performance. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*.

Vogel, A., & Jurafsky, D. (2010). Learning to follow navigational directions. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Wong, Y., & Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the annual meeting of the North American association of computational linguistics (NAACL)*.

Wong, Y., & Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.

Yu, C., & Joachims, T. (2009). Learning structural svms with latent variables. In *Proceedings of the international conference on machine learning (ICML)*.

Zelle, J., & Mooney, R. (1996). Learning to parse database queries using inductive logic programing. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*.

Zettlemoyer, L., & Collins, M. (2005). Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the conference on uncertainty in artificial intelligence (UAI)*.

Zettlemoyer, L., & Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the conference on computational natural language learning (CoNLL)*.

Zettlemoyer, L., & Collins, M. (2009). Learning context-dependent mappings from sentences to logical form. In *Proceedings of the annual meeting of the association for computational linguistics (ACL)*.