Learning Context-Dependent Mappings from Sentences to Logical Form

Luke S. Zettlemoyer and **Michael Collins**
MIT CSAIL
Cambridge, MA 02139
{lsz,mcollins}@csail.mit.com

Abstract

We consider the problem of learning context-dependent mappings from sentences to logical form. The training examples are sequences of sentences annotated with lambda-calculus meaning representations. We develop an algorithm that maintains explicit, lambda-calculus representations of salient discourse entities and uses a context-dependent analysis pipeline to recover logical forms. The method uses a hidden-variable variant of the perception algorithm to learn a linear model used to select the best analysis. Experiments on context-dependent utterances from the ATIS corpus show that the method recovers fully correct logical forms with 83.7% accuracy.

1 Introduction

Recently, researchers have developed algorithms that learn to map natural language sentences to representations of their underlying meaning (He and Young, 2006; Wong and Mooney, 2007; Zettlemoyer and Collins, 2005). For instance, a training example might be:

Sent. 1: List flights to Boston on Friday night.
LF 1: $\lambda x.flight(x) \wedge to(x, bos)$
$\wedge\, day(x, fri) \wedge during(x, night)$

Here the logical form (LF) is a lambda-calculus expression defining a set of entities that are flights to Boston departing on Friday night.

Most of this work has focused on analyzing sentences in isolation. In this paper, we consider the problem of learning to interpret sentences whose underlying meanings can depend on the context in which they appear. For example, consider an interaction where Sent. 1 is followed by the sentence:

Sent. 2: Show me the flights after 3pm.
LF 2: $\lambda x.flight(x) \wedge to(x, bos)$
$\wedge day(x, fri) \wedge depart(x) > 1500$

In this case, the fact that Sent. 2 describes flights to Boston on Friday must be determined based on the context established by the first sentence.

We introduce a supervised, hidden-variable approach for learning to interpret sentences in context. Each training example is a sequence of sentences annotated with logical forms. Figure 1 shows excerpts from three training examples in the ATIS corpus (Dahl et al., 1994).

For context-dependent analysis, we develop an approach that maintains explicit, lambda-calculus representations of salient discourse entities and uses a two-stage pipeline to construct context-dependent logical forms. The first stage uses a probabilistic Combinatory Categorial Grammar (CCG) parsing algorithm to produce a context-independent, underspecified meaning representation. The second stage resolves this underspecified meaning representation by making a sequence of modifications to it that depend on the context provided by previous utterances.

In general, there are a large number of possible context-dependent analyses for each sentence. To select the best one, we present a weighted linear model that is used to make a range of parsing and context-resolution decisions. Since the training data contains only the final logical forms, we model these intermediate decisions as hidden variables that must be estimated without explicit supervision. We show that this model can be effectively trained with a hidden-variable variant of the perceptron algorithm.

In experiments on the ATIS DEC94 test set, the approach recovers fully correct logical forms with 83.7% accuracy.

2 The Learning Problem

We assume access to a training set that consists of n *interactions* $D = \langle I_1, \ldots, I_n \rangle$. The i'th interaction I_i contains n_i sentences, $w_{i,1}, \ldots, w_{i,n_i}$. Each sentence $w_{i,j}$ is paired with a lambda-calculus ex-

Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 976–984,
Suntec, Singapore, 2-7 August 2009.

Example #1:
(a) show me the flights from boston to philly
$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$
(b) show me the ones that leave in the morning
$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$
$\wedge during(x, morning)$
(c) what kind of plane is used on these flights
$\lambda y.\exists x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$
$\wedge during(x, morning) \wedge aircraft(x) = y$

Example #2:
(a) show me flights from milwaukee to orlando
$\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl)$
(b) cheapest
$argmin(\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl),$
$\lambda y.fare(y))$
(c) departing wednesday after 5 o'clock
$argmin(\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl)$
$\wedge day(x, wed) \wedge depart(x) > 1700$,
$\lambda y.fare(y))$

Example #3:
(a) show me flights from pittsburgh to la thursday evening
$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$
$\wedge day(x, thur) \wedge during(x, evening)$
(b) thursday afternoon
$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$
$\wedge day(x, thur) \wedge during(x, afternoon)$
(c) thursday after 1700 hours
$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$
$\wedge day(x, thur) \wedge depart(x) > 1700$

Figure 1: ATIS interaction excerpts.

pression $z_{i,j}$ specifying the target logical form. Figure 1 contains example interactions.

The logical forms in the training set are representations of each sentence's underlying meaning. In most cases, context (the previous utterances and their interpretations) is required to recover the logical form for a sentence. For instance, in Example 1(b) in Figure 1, the sentence "show me the ones that leave in the morning" is paired with

$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$
$\wedge during(x, morning)$

Some parts of this logical form ($from(x, bos)$ and $to(x, phi)$) depend on the context. They have to be recovered from the previous logical forms.

At step j in interaction i, we define the *context* $\langle z_{i,1}, \ldots, z_{i,j-1} \rangle$ to be the $j-1$ preceding logical forms.[1] Now, given the training data, we can create training examples $(x_{i,j}, z_{i,j})$ for $i = 1 \ldots n, j = 1 \ldots n_i$. Each $x_{i,j}$ is a sentence and a context, $x_{i,j} = (w_{i,j}, \langle z_{i,1}, \ldots, z_{i,j-1} \rangle)$. Given this set up, we have a supervised learning problem with input $x_{i,j}$ and output $z_{i,j}$.

[1] In general, the context could also include the previous sentences $w_{i,k}$ for $k < j$. In our data, we never observed any interactions where the choice of the correct logical form $z_{i,j}$ depended on the words in the previous sentences.

3 Overview of Approach

In general, the mapping from a sentence and a context to a logical form can be quite complex. In this section, we present an overview of our learning approach. We assume the learning algorithm has access to:

- A training set D, defined in Section 2.
- A CCG lexicon.[2] See Section 4 for an overview of CCG. Each entry in the lexicon pairs a word (or sequence of words), with a CCG category specifying both the syntax and semantics for that word. One example CCG entry would pair *flights* with the category $N : \lambda x.flight(x)$.

Derivations A *derivation* for the j'th sentence in an interaction takes as input a pair $x = (w_j, C)$, where $C = \langle z_1 \ldots z_{j-1} \rangle$ is the current context. It produces a logical form z. There are two stages:

- First, the sentence w_j is parsed using the CCG lexicon to form an intermediate, context-independent logical form π.
- Second, in a series of steps, π is mapped to z. These steps depend on the context C.

As one sketch of a derivation, consider how we might analyze Example 1(b) in Figure 1. In this case the sentence is "show me the ones that leave in the morning." The CCG parser would produce the following context-independent logical form:

$\lambda x.!\langle e, t \rangle(x) \wedge during(x, morning)$

The subexpression $!\langle e, t \rangle$ results directly from the referential phrase *the ones*; we discuss this in more detail in Section 4.2, but intuitively this subexpression specifies that a lambda-calculus expression of type $\langle e, t \rangle$ must be recovered from the context and substituted in its place.

In the second (contextually dependent) stage of the derivation, the expression

$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$

is recovered from the context, and substituted for the $!\langle e, t \rangle$ subexpression, producing the desired final logical form, seen in Example 1(b).

[2] Developing algorithms that learn the CCG lexicon from the data described in this paper is an important area for future work. We could possibly extend algorithms that learn from context-independent data (Zettlemoyer and Collins, 2005).

In addition to substitutions of this type, we will also perform other types of context-dependent resolution steps, as described in Section 5.

In general, both of the stages of the derivation involve considerable ambiguity – there will be a large number of possible context-independent logical forms π for w_j and many ways of modifying each π to create a final logical form z_j.

Learning We model the problem of selecting the best derivation as a structured prediction problem (Johnson et al., 1999; Lafferty et al., 2001; Collins, 2002; Taskar et al., 2004). We present a linear model with features for both the parsing and context resolution stages of the derivation. In our setting, the choice of the context-independent logical form π and all of the steps that map π to the output z are hidden variables; these steps are not annotated in the training data. To estimate the parameters of the model, we use a hidden-variable version of the perceptron algorithm. We use an approximate search procedure to find the best derivation both while training the model and while applying it to test examples.

Evaluation We evaluate the approach on sequences of sentences $\langle w_1, \ldots, w_k \rangle$. For each w_j, the algorithm constructs an output logical form z_j which is compared to a gold standard annotation to check correctness. At step j, the context contains the previous z_i, for $i < j$, output by the system.

4 Context-independent Parsing

In this section, we first briefly review the CCG parsing formalism. We then define a set of extensions that allow the parser to construct logical forms containing references, such as the $!\langle e, t \rangle$ expression from the example derivation in Section 3.

4.1 Background: CCG

CCG is a lexicalized, mildly context-sensitive parsing formalism that models a wide range of linguistic phenomena (Steedman, 1996; Steedman, 2000). Parses are constructed by combining lexical entries according to a small set of relatively simple rules. For example, consider the lexicon

$$\begin{array}{lcl} \text{flights} & := & N : \lambda x.flight(x) \\ \text{to} & := & (N \backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x, y) \\ \text{boston} & := & NP : boston \end{array}$$

Each lexical entry consists of a word and a category. Each category includes syntactic and semantic content. For example, the first entry pairs the word *flights* with the category $N : \lambda x.flight(x)$. This category has syntactic type N, and includes the lambda-calculus semantic expression $\lambda x.flight(x)$. In general, syntactic types can either be simple types such as N, NP, or S, or can be more complex types that make use of slash notation, for example $(N \backslash N)/NP$.

CCG parses construct parse trees according to a set of *combinator* rules. For example, consider the *functional application* combinators:[3]

$$\begin{array}{llcll} A/B : f & B : g & \Rightarrow & A : f(g) & (>) \\ B : g & A \backslash B : f & \Rightarrow & A : f(g) & (<) \end{array}$$

The first rule is used to combine a category with syntactic type A/B with a category to the right of syntactic type B to create a new category of type A. It also constructs a new lambda-calculus expression by applying the function f to the expression g. The second rule handles arguments to the left. Using these rules, we can parse the following phrase:

$$\dfrac{\dfrac{\dfrac{\text{flights}}{\begin{array}{c} N \\ \lambda x.flight(x) \end{array}} \quad \dfrac{\dfrac{\text{to}}{\begin{array}{c} (N \backslash N)/NP \\ \lambda y.\lambda f.\lambda x.f(x) \wedge to(x, y) \end{array}} \quad \dfrac{\text{boston}}{\begin{array}{c} NP \\ boston \end{array}}}{\begin{array}{c} (N \backslash N) \\ \lambda f.\lambda x.f(x) \wedge to(x, boston) \end{array}}}{\begin{array}{c} N \\ \lambda x.flight(x) \wedge to(x, boston) \end{array}}$$

The top-most parse operations pair each word with a corresponding category from the lexicon. The later steps are labeled with the rule that was applied ($>$ for the first and $<$ for the second).

4.2 Parsing with References

In this section, we extend the CCG parser to introduce references. We use an exclamation point followed by a type expression to specify references in a logical form. For example, $!e$ is a reference to an entity and $!\langle e, t \rangle$ is a reference to a function. As motivated in Section 3, we introduce these expressions so they can later be replaced with appropriate lambda-calculus expressions from the context.

Sometimes references are lexically triggered. For example, consider parsing the phrase "show me the ones that leave in the morning" from Example 1(b) in Figure 1. Given the lexical entry:

$$\text{ones} \quad := \quad N : \lambda x.!\langle e, t \rangle(x)$$

a CCG parser could produce the desired context-

[3] In addition to application, we make use of composition, type raising and coordination combinators. A full description of these combinators is beyond the scope of this paper. Steedman (1996; 2000) presents a detailed description of CCG.

independent logical form:

$$\lambda x.!\langle e,t\rangle(x) \wedge during(x, morning)$$

Our first extension is to simply introduce lexical items that include references into the CCG lexicon. They describe anaphoric words, for example including "ones," "those," and "it."

In addition, we sometimes need to introduce references when there is no explicit lexical trigger. For instance, Example 2(c) in Figure 1 consists of the single word "cheapest." This query has the same meaning as the longer request "show me the cheapest one," but it does not include the lexical reference. We add three CCG type-shifting rules to handle these cases.

The first two new rules are applicable when there is a category that is expecting an argument with type $\langle e,t\rangle$. This argument is replaced with a $!\langle e,t\rangle$ reference:

$$A/B : f \quad \Rightarrow \quad A : f(\lambda x.!\langle e,t\rangle(x))$$
$$A\backslash B : f \quad \Rightarrow \quad A : f(\lambda x.!\langle e,t\rangle(x))$$

For example, using the first rule, we could produce the following parse for Example 2(c)

$$\dfrac{\dfrac{\text{cheapest}}{\dfrac{NP/N}{\lambda g.argmin(\lambda x.g(x), \lambda y.fare(y))}}}{\dfrac{NP}{argmin(\lambda x.!\langle e,t\rangle(x), \lambda y.fare(y))}}$$

where the final category has the desired lambda-caculus expression.

The third rule is motivated by examples such as "show me nonstop flights." Consider this sentence being uttered after Example 1(a) in Figure 1. Although there is a complete, context-independent meaning, the request actually restricts the salient set of flights to include only the nonstop ones. To achieve this analysis, we introduce the rule:

$$A : f \quad \Rightarrow \quad A : \lambda x.f(x) \ \wedge \ !\langle e,t\rangle(x)$$

where f is an function of type $\langle e,t\rangle$.

With this rule, we can construct the parse

$$\dfrac{\dfrac{\dfrac{\text{nonstop}}{\dfrac{N/N}{\lambda f.\lambda x.f(x) \wedge nonstop(x)}} \quad \dfrac{\text{flights}}{\dfrac{N}{\lambda x.flight(x)}}}{\dfrac{N}{\lambda x.nonstop(x) \wedge flight(x)}}}{\dfrac{N}{\lambda x.nonstop(x) \wedge flight(x) \ \wedge \ !\langle e,t\rangle(x)}}$$

where the last parsing step is achieved with the new type-shifting rule.

These three new parsing rules allow significant flexibility when introducing references. Later, we develop an approach that learns when to introduce references and how to best resolve them.

5 Contextual Analysis

In this section, we first introduce the general patterns of context-dependent analysis that we consider. We then formally define derivations that model these phenomena.

5.1 Overview

This section presents an overview of the ways that the context C is used during the analysis.

References Every reference expression ($!e$ or $!\langle e,t\rangle$) must be replaced with an expression from the context. For example, in Section 3, we considered the following logical form:

$$\lambda x.!\langle e,t\rangle(x) \wedge during(x, morning)$$

In this case, we saw that replacing the $!\langle e,t\rangle$ subexpression with the logical form for Example 1(a), which is directly available in C, produces the desired final meaning.

Elaborations Later statements can expand the meaning of previous ones in ways that are difficult to model with references. For example, consider analyzing Example 2(c) in Figure 1. Here the phrase "departing wednesday after 5 o'clock" has a context-independent logical form:[4]

$$\lambda x.day(x, wed) \wedge depart(x) > 1700 \qquad (1)$$

that must be combined with the meaning of the previous sentence from the context C:

$$argmin(\lambda x.fight(x) \wedge from(x, mil) \wedge to(x, orl), \lambda y.fare(y))$$

to produce the expression

$$argmin(\lambda x.fight(x) \wedge from(x, mil) \wedge to(x, orl) \wedge day(x, wed) \wedge depart(x) > 1700, \lambda y.fare(y))$$

Intuitively, the phrase "departing wednesday after 5 o'clock" is providing new constraints for the set of flights embedded in the $argmin$ expression.

We handle examples of this type by constructing *elaboration expressions* from the z_i in C. For example, if we constructed the following function:

$$\lambda f.argmin(\lambda x.fight(x) \wedge from(x, mil) \wedge to(x, orl) \wedge f(x), \lambda y.fare(y)) \qquad (2)$$

[4] Another possible option is the expression $\lambda x.!\langle e,t\rangle \wedge day(x, wed) \wedge depart(x) > 1700$. However, there is no obvious way to resolve the $!\langle e,t\rangle$ expression that would produce the desired final meaning.

we could apply this function to Expression 1 and produce the desired result. The introduction of the new variable f provides a mechanism for expanding the embedded subexpression.

References with Deletion When resolving references, we will sometimes need to delete subparts of the expressions that we substitute from the context. For instance, consider Example 3(b) in Figure 1. The desired, final logical form is:

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \\ \wedge day(x, thur) \wedge during(x, afternoon)$$

We need to construct this from the context-independent logical form:

$$\lambda x.!\langle e, t\rangle \wedge day(x, thur) \wedge during(x, afternoon)$$

The reference $!\langle e, t\rangle$ must be resolved. The only expression in the context C is the meaning from the previous sentence, Example 3(a):

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \\ \wedge day(x, thur) \wedge during(x, evening) \quad (3)$$

Substituting this expression directly would produce the following logical form:

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \\ \wedge day(x, thur) \wedge during(x, evening) \\ \wedge day(x, thur) \wedge during(x, afternoon)$$

which specifies the day twice and has two different time spans.

We can achieve the desired analysis by deleting parts of expressions before they are substituted. For example, we could remove the day and time constraints from Expression 3 to create:

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$$

which would produce the desired final meaning when substituted into the original expression.

Elaborations with Deletion We also allow deletions for elaborations. In this case, we delete subexpressions of the elaboration expression that is constructed from the context.

5.2 Derivations

We now formally define a *derivation* that maps a sentence w_j and a context $C = \{z_1, \ldots, z_{j-1}\}$ to an output logical form z_j. We first introduce notation for expressions in C that we will use in the derivation steps. We then present a definition of deletion. Finally, we define complete derivations.

Context Sets Given a context C, our algorithm constructs three sets of expressions:

- $R_e(C)$: A set of e-type expressions that can be used to resolve references.
- $R_{\langle e,t\rangle}(C)$: A set of $\langle e, t\rangle$-type expressions that can be used to resolve references.
- $E(C)$: A set of possible elaboration expressions (for example, see Expression 2).

We will provide the details of how these sets are defined in Section 5.3. As an example, if C contains only the logical form

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$$

then $R_e(C) = \{pit, la\}$ and $R_{\langle e,t\rangle}(C)$ is a set that contains a single entry, the complete logical form.

Deletion A deletion operator accepts a logical form l and produces a new logical form l'. It constructs l' by removing a single subexpression that appears in a coordination (conjunction or disjunction) in l. For example, if l is

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$$

there are three possible deletion operations, each of which removes a single subexpression.

Derivations We now formally define a *derivation* to be a sequence $d = (\Pi, s_1, \ldots, s_m)$. Π is a CCG parse that constructs a context-independent logical form π with $m - 1$ reference expressions.[5] Each s_i is a function that accepts as input a logical form, makes some change to it, and produces a new logical form that is input to the next function s_{i+1}. The initial s_i for $i < m$ are *reference steps*. The final s_m is an optional *elaboration step*.

- **Reference Steps:** A reference step is a tuple $(l, l', f, r, r_1, \ldots, r_p)$. This operator selects a reference f in the input logical form l and an appropriately typed expression r from either $R_e(C)$ or $R_{\langle e,t\rangle}(C)$. It then applies a sequence of p deletion operators to create new expressions $r_1 \ldots r_p$. Finally, it constructs the output logical form l' by substituting r_p for the selected reference f in l.
- **Elaboration Steps:** An elaboration step is a tuple $(l, l', b, b_1, \ldots, b_q)$. This operator selects an expression b from $E(C)$ and applies q deletions to create new expressions $b_1 \ldots b_q$. The output expression l' is $b_q(l)$.

[5] In practice, π rarely contains more than one reference.

In general, the space of possible derivations is large. In Section 6, we describe a linear model and decoding algorithm that we use to find high scoring derivations.

5.3 Context Sets

For a context $C = \{z_1, \ldots, z_{j-1}\}$, we define sets $R_e(C)$, $R_{\langle e,t\rangle}(C)$, and $E(C)$ as follows.

e-type Expressions $R_e(z)$ is a set of e-type expressions extracted from a logical form z. We define $R_e(C) = \bigcup_{i=1}^{j-1} R_e(z_i)$.

$R_e(z)$ includes all e-type subexpressions of z.[6] For example, if z is

$$argmin(\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl), \lambda y.fare(y))$$

the resulting set is $R_e(z) = \{mil, orl, z\}$, where z is included because the entire $argmin$ expression has type e.

$\langle e,t\rangle$-type Expressions $R_{\langle e,t\rangle}(z)$ is a set of $\langle e,t\rangle$-type expressions extracted from a logical form z. We define $R_{\langle e,t\rangle}(C) = \bigcup_{i=1}^{j-1} R_{\langle e,t\rangle}(z_i)$.

The set $R_{\langle e,t\rangle}(z)$ contains all of the $\langle e,t\rangle$-type subexpressions of z. For each quantified variable x in z, it also contains a function $\lambda x.g$. The expression g contains the subexpressions in the scope of x that do not have free variables. For example, if z is

$$\lambda y.\exists x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning) \wedge aircraft(x) = y$$

$R_{\langle e,t\rangle}(z)$ would contain two functions: the entire expression z and the function

$$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning)$$

constructed from the variable x, where the subexpression $aircraft(x) = y$ has been removed because it contains the free variable y.

Elaboration Expressions Finally, $E(z)$ is a set of elaboration expressions constructed from a logical form z. We define $E(C) = \bigcup_{i=1}^{j-1} E(z_i)$.

$E(z)$ is defined by enumerating the places where embedded variables are found in z. For each logical variable x and each coordination (conjunction or disjunction) in the scope of x, a new expression is created by defining a function $\lambda f.z'$ where z' has the function $f(x)$ added to the appropriate coordination. This procedure would produce the example elaboration Expression 2 and elaborations that expand other embedded expressions, such as the quantifier in Example 1(c).

6 A Linear Model

In general, there will be many possible derivations d for an input sentence w in the current context C. In this section, we introduce a weighted linear model that scores derivations and a decoding algorithm that finds high scoring analyses.

We define $\text{GEN}(w; C)$ to be the set of possible derivations d for an input sentence w given a context C, as described in Section 5.2. Let $\phi(d) \in \mathbb{R}^m$ be an m-dimensional feature representation for a derivation d and $\theta \in \mathbb{R}^m$ be an m-dimensional parameter vector. The optimal derivation for a sentence w given context C and parameters θ is

$$d^*(w; C) = \arg \max_{d \in \text{GEN}(w;C)} \theta \cdot \phi(d)$$

Decoding We now describe an approximate algorithm for computing $d^*(w; C)$.

The CCG parser uses a CKY-style chart parsing algorithm that prunes to the top $N = 50$ entries for each span in the chart.

We use a beam search procedure to find the best contextual derivations, with beam size $N = 50$. The beam is initialized to the top N logical forms from the CCG parser. The derivations are extended with reference and elaboration steps. The only complication is selecting the sequence of deletions. For each possible step, we use a greedy search procedure that selects the sequence of deletions that would maximize the score of the derivation after the step is applied.

7 Learning

Figure 2 details the complete learning algorithm. Training is online and error-driven. Step 1 parses the current sentence in context. If the optimal logical form is not correct, Step 2 finds the best derivation that produces the labeled logical form[7] and does an additive, perceptron-style parameter update. Step 3 updates the context. This algorithm is a direct extension of the one introduced by Zettlemoyer and Collins (2007). It maintains the context but does not have the lexical induction step that was previously used.

[6] A lambda-calculus expression can be represented as a tree structure with flat branching for coordination (conjunction and disjunction). The subexpressions are the subtrees.

[7] For this computation, we use a modified version of the beam search algorithm described in Section 6, which prunes derivations that could not produce the desired logical form.

Inputs: Training examples $\{I_i | i = 1 \dots n\}$. Each I_i is a sequence $\{(w_{i,j}, z_{i,j}) : j = 1 \dots n_i\}$ where $w_{i,j}$ is a sentence and $z_{i,j}$ is a logical form. Number of training iterations T. Initial parameters θ.

Definitions: The function $\phi(d)$ represents the features described in Section 8. $\text{GEN}(w; C)$ is the set of derivations for sentence w in context C. $\text{GEN}(w, z; C)$ is the set of derivations for sentence w in context C that produce the final logical form z. The function $L(d)$ maps a derivation to its associated final logical form.

Algorithm:

- For $t = 1 \dots T, i = 1 \dots n$: (Iterate interactions)
  - Set $C = \{\}$. (Reset context)
  - For $j = 1 \dots n_i$: (Iterate training examples)

    Step 1: (Check correctness)
    - Let $d^* = \arg\max_{d \in \text{GEN}(w_{i,j}; C)} \theta \cdot \phi(d)$.
    - If $L(d^*) = z_{i,j}$, go to Step 3.

    Step 2: (Update parameters)
    - Let $d' = \arg\max_{d \in \text{GEN}(w_{i,j}, z_{i,j}; C)} \theta \cdot \phi(d)$.
    - Set $\theta = \theta + \phi(d') - \phi(d^*)$.

    Step 3: (Update context)
    - Append $z_{i,j}$ to the current context C.

Output: Estimated parameters θ.

Figure 2: An online learning algorithm.

8 Features

We now describe the features for both the parsing and context resolution stages of the derivation.

8.1 Parsing Features

The parsing features are used to score the context-independent CCG parses during the first stage of analysis. We use the set developed by Zettlemoyer and Collins (2007), which includes features that are sensitive to lexical choices and the structure of the logical form that is constructed.

8.2 Context Features

The context features are functions of the derivation steps described in Section 5.2. In a derivation for sentence j of an interaction, let l be the input logical form when considering a new step s (a reference or elaboration step). Let c be the expression that s selects from a context set $R_e(z_i)$, $R_{\langle e,t \rangle}(z_i)$, or $E(z_i)$, where z_i, $i < j$, is an expression in the current context. Also, let r be a subexpression deleted from c. Finally, let f_1 and f_2 be predicates, for example $from$ or to.

Distance Features The distance features are binary indicators on the distance $j - i$. These features allow the model to, for example, favor resolving references with lambda-calculus expressions recovered from recent sentences.

Copy Features For each possible f_1 there is a feature that tests if f_1 is present in the context expression c but not in the current expression l. These features allow the model to learn to select expressions from the context that introduce expected predicates. For example, flights usually have a $from$ predicate in the current expression.

Deletion Features For each pair (f_1, f_2) there is a feature that tests if f_1 is in the current expression l and f_2 is in the deleted expression r. For example, if $f_1 = f_2 = days$ the model can favor overriding old constraints about the departure day with new ones introduced in the current utterance. When $f_1 = during$ and $f_2 = depart\_time$ the algorithm can learn that specific constraints on the departure time override more general constraints about the period of day.

9 Related Work

There has been a significant amount of work on the problem of learning context-independent mappings from sentences to meaning representations. Researchers have developed approaches using models and algorithms from statistical machine translation (Papineni et al., 1997; Ramaswamy and Kleindienst, 2000; Wong and Mooney, 2007), statistical parsing (Miller et al., 1996; Ge and Mooney, 2005), inductive logic programming (Zelle and Mooney, 1996; Tang and Mooney, 2000) and probabilistic push-down automata (He and Young, 2006).

There were a large number of successful hand-engineered systems developed for the original ATIS task and other related tasks (e.g., (Carbonell and Hayes, 1983; Seneff, 1992; Ward and Issar, 1994; Levin et al., 2000; Popescu et al., 2004)). We are only aware of one system that learns to construct context-dependent interpretations (Miller et al., 1996). The Miller et al. (1996) approach is fully supervised and produces a final meaning representation in SQL. It requires complete annotation of all of the syntactic, semantic, and discourse decisions required to correctly analyze each training example. In contrast, we learn from examples annotated with lambda-calculus expressions that represent only the final, context-dependent logical forms.

Finally, the CCG (Steedman, 1996; Steedman,

	Train	Dev.	Test	All
Interactions	300	99	127	526
Sentences	2956	857	826	4637

Table 1: Statistics of the ATIS training, development and test (DEC94) sets, including the total number of interactions and sentences. Each interaction is a sequence of sentences.

2000) parsing setup is closely related to previous CCG research, including work on learning parsing models (Clark and Curran, 2003), wide-coverage semantic parsing (Bos et al., 2004) and grammar induction (Watkinson and Manandhar, 1999).

10 Evaluation

Data In this section, we present experiments in the context-dependent ATIS domain (Dahl et al., 1994). Table 1 presents statistics for the training, development, and test sets. To facilitate comparison with previous work, we used the standard DEC94 test set. We randomly split the remaining data to make training and development sets. We manually converted the original SQL meaning annotations to lambda-calculus expressions.

Evaluation Metrics Miller et al. (1996) report accuracy rates for recovering correct SQL annotations on the test set. For comparison, we report *exact accuracy* rates for recovering completely correct lambda-calculus expressions.

We also present precision, recall and F-measure for *partial match* results that test if individual attributes, such as the from and to cities, are correctly assigned. See the discussion by Zettlemoyer and Collins (2007) (ZC07) for the full details.

Initialization and Parameters The CCG lexicon is hand engineered. We constructed it by running the ZC07 algorithm to learn a lexicon on the context-independent ATIS data set and making manual corrections to improve performance on the training set. We also added lexical items with reference expressions, as described in Section 4.

We ran the learning algorithm for $T = 4$ training iterations. The parsing feature weights were initialized as in ZC07, the context distance features were given small negative weights, and all other feature weights were initially set to zero.

Test Setup During evaluation, the context $C = \{z_1 \dots z_{j-1}\}$ contains the logical forms output by the learned system for the previous sentences. In general, errors made while constructing these expressions can propogate if they are used in derivations for new sentences.

System	Partial Match			Exact
	Prec.	Rec.	F1	Acc.
Full Method	95.0	96.5	95.7	83.7
Miller et al.	–	–	–	78.4

Table 2: Performance on the ATIS DEC94 test set.

Limited Context	Partial Match			Exact
	Prec.	Rec.	F1	Acc.
$M = 0$	96.2	57.3	71.8	45.4
$M = 1$	94.9	91.6	93.2	79.8
$M = 2$	94.8	93.2	94.0	81.0
$M = 3$	94.5	94.3	94.4	82.1
$M = 4$	94.9	92.9	93.9	81.6
$M = 10$	94.2	94.0	94.1	81.4

Table 3: Performance on the ATIS development set for varying context window lengths M.

Results Table 2 shows performance on the ATIS DEC94 test set. Our approach correctly recovers 83.7% of the logical forms. This result compares favorably to Miller et al.'s fully-supervised approach (1996) while requiring significantly less annotation effort.

We also evaluated performance when the context is limited to contain only the M most recent logical forms. Table 3 shows results on the development set for different values of M. The poor performance with no context ($M = 0$) demonstrates the need for context-dependent analysis. Limiting the context to the most recent statement ($M = 1$) significantly improves performance while using the last three utterances ($M = 3$) provides the best results.

Finally, we evaluated a variation where the context contains gold-standard logical forms during evaluation instead of the output of the learned model. On the development set, this approach achieved 85.5% exact-match accuracy, an improvement of approximately 3% over the standard approach. This result suggests that incorrect logical forms in the context have a relatively limited impact on overall performance.

11 Conclusion

In this paper, we addressed the problem of learning context-dependent mappings from sentences to logical form. We developed a context-dependent analysis model and showed that it can be effectively trained with a hidden-variable variant of the perceptron algorithm. In the experiments, we showed that the approach recovers fully correct logical forms with 83.7% accuracy.

References

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the International Conference on Computational Linguistics*.

Jaime G. Carbonell and Philip J. Hayes. 1983. Recovery strategies for parsing extragrammatical language. *American Journal of Computational Linguistics*, 9.

Stephen Clark and James R. Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Michael Collins. 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.*

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: the ATIS-3 corpus. In *ARPA HLT Workshop*.

Ruifang Ge and Raymond J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Conference on Computational Natural Language Learning*.

Yulan He and Steve Young. 2006. Spoken language understanding using the hidden vector state model. *Speech Communication*, 48(3-4).

Mark Johnson, Stuart Geman, Steven Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proc. of the Association for Computational Linguistics*.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.

E. Levin, S. Narayanan, R. Pieraccini, K. Biatov, E. Bocchieri, G. Di Fabbrizio, W. Eckert, S. Lee, A. Pokrovsky, M. Rahim, P. Ruscitti, and M. Walker. 2000. The AT&T darpa communicator mixed-initiative spoken dialogue system. In *Proceedings of the International Conference on Spoken Language Processing*.

Scott Miller, David Stallard, Robert J. Bobrow, and Richard L. Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proc. of the Association for Computational Linguistics*.

K. A. Papineni, S. Roukos, and T. R. Ward. 1997. Feature-based language understanding. In *Proceedings of European Conference on Speech Communication and Technology*.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the International Conference on Computational Linguistics*.

Ganesh N. Ramaswamy and Jan Kleindienst. 2000. Hierarchical feature-based translation for scalable natural language understanding. In *Proceedings of International Conference on Spoken Language Processing*.

Stephanie Seneff. 1992. Robust parsing for spoken language systems. In *Proc. of the IEEE Conference on Acoustics, Speech, and Signal Processing*.

Mark Steedman. 1996. *Surface Structure and Interpretation*. The MIT Press.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press.

Lappoon R. Tang and Raymond J. Mooney. 2000. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Wayne Ward and Sunil Issar. 1994. Recent improvements in the CMU spoken language understanding system. In *Proceedings of the workshop on Human Language Technology*.

Stephen Watkinson and Suresh Manandhar. 1999. Unsupervised lexical learning with categorial grammars using the LLL corpus. In *Proceedings of the 1st Workshop on Learning Language in Logic*.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Association for Computational Linguistics*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proc. of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.