Learning Context-Dependent Mappings from Sentences to Logical Form


As Published
Publisher: Association for Computing Machinery

Version: Author’s final manuscript

Accessed: Fri Feb 08 14:55:52 EST 2019

Citable Link: http://hdl.handle.net/1721.1/59354

Terms of Use: Attribution-Noncommercial-Share Alike 3.0 Unported

Detailed Terms: http://creativecommons.org/licenses/by-nc-sa/3.0/
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 perceptron 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: \( \lambda x.\text{flight}(x) \land \text{to}(x, \text{bos}) \land \text{day}(x, \text{fri}) \land \text{during}(x, \text{night}) \)

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: \( \lambda x.\text{flight}(x) \land \text{to}(x, \text{bos}) \land \text{day}(x, \text{fri}) \land \text{depart}(x) > 1500 \)

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

Example #2:
(a) show me flights from milwaukee to orlando
\[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}) \]
(b) cheapest
\[ \text{argmin}(\lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}) \land \text{day}(x, \text{wed}) \land \text{depart}(x) > 1700, \lambda y. \text{fare}(y)) \]
(c) departing wednesday after 5 o’clock
\[ \text{argmin}(\lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}) \land \text{day}(x, \text{wed}) \land \text{depart}(x) > 1700, \lambda x, \text{thur} \land \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}) \land \text{day}(x, \text{morning}) \land \text{aircraft}(z) = y) \]

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

Figure 1: ATIS interaction excerpts.

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
\[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \text{phi}) \land \text{during}(x, \text{morning}) \]

Some parts of this logical form (\text{from}(x, \text{bos}) and \text{to}(x, \text{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 \( z_{i,j} \).

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 \text{flights} with the category \( N : \lambda x. \text{flight}(x) \).

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 \( \pi \). There are two stages:

- First, the sentence \( w_j \) is parsed using the CCG lexicon to form an intermediate, context-independent logical form \( \pi \).
- Second, in a series of steps, \( \pi \) 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:
\[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \text{phi}) \land \text{during}(x, \text{morning}) \]

The subexpression \( !\langle e, t \rangle \) results directly from the referential phrase \text{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
\[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \text{phi}) \land \text{during}(x, \text{morning}) \]
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).

\(^1\)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.

\(^2\)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 \( \pi \) for \( w_j \) and many ways of modifying each \( \pi \) 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 \( \pi \) and all of the steps that map \( \pi \) 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 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 exclamation 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

\[
\begin{align*}
\text{flights} & : N : \lambda x. \text{flight}(x) \\
\text{to} & : (N \setminus N)/NP : \lambda y. \lambda f. \lambda x. f(x) \land \text{to}(x, y) \\
\text{boston} & : N/P : \lambda x. \text{flight}(x) \land \text{to}(x, \text{boston})
\end{align*}
\]

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 \( \text{flights} \) with the category \( N : \lambda x. \text{flight}(x) \). This category has syntactic type \( N \), and includes the lambda-calculus semantic expression \( \lambda x. \text{flight}(x) \). 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 \( (N \setminus N)/NP \).

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

\[
\begin{align*}
A/B : f \quad B : g & \Rightarrow A : f(g) \quad (>) \\
B : g \quad A/B : f & \Rightarrow A : f(g) \quad (<)
\end{align*}
\]

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:

\[
\begin{align*}
\text{flights} & : N : \lambda x. \text{flight}(x) \\
\text{to} & : (N \setminus N)/NP : \lambda y. \lambda f. \lambda x. f(x) \land \text{to}(x, y) \\
\text{boston} & : N/P : \lambda x. \text{flight}(x) \land \text{to}(x, \text{boston})
\end{align*}
\]

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 (\( \Rightarrow \) for the first and \( \Leftarrow \) 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:

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

a CCG parser could produce the desired context-
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 type \(\langle e,t\rangle\). This argument is replaced with a !\(\langle e,t\rangle\) reference:

\[
A/B : f \Rightarrow A : f(\lambda x.!(\langle e,t\rangle(x))
\]

\[
A\setminus B : f \Rightarrow A : f(\lambda x.!(\langle e,t\rangle(x))
\]

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

```
NP/N
λg.argmin(λx.g(x), λy.fare(y))
```

where the final category has the desired lambda-calcus 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:

\[
A : f \Rightarrow A : \lambda x.f(x) \land !(\lambda e,t)(x)
\]

where \(f\) is a function of type \(\langle e,t\rangle\).

With this rule, we can construct the parse

```
NP/N
\[
λf.λx.f(x) \land \text{nonstop}(x)
\]
```

```
NP/N
\[
\text{flights}(x)
\]
```

```
NP/N
\[
\lambda x.\text{flight}(x)
\]
```

```
NP/N
\[
\lambda x.\text{nonstop}(x) \land \text{flight}(x)
\]
```

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

\[
\lambda x.!(\langle e,t\rangle(x) \land \text{during}(x, \text{morning}))
\]

In this case, we saw that replacing the \(!!(e,t)\) 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:

\[
\lambda x.\text{day}(x, \text{wed}) \land \text{depart}(x) > 1700
\]

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

\[
\text{argmin}(\lambda x.\text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{ori}), \lambda y.\text{fare}(y))
\]

to produce the expression

\[
\text{argmin}(\lambda x.\text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{ori}) \land \text{day}(x, \text{wed}) \land \text{depart}(x) > 1700, \lambda y.\text{fare}(y))
\]

Intuitively, the phrase “departing wednesday after 5 o’clock” is providing new constraints for the set of flights embedded in the \text{argmin} expression.

We handle examples of this type by constructing \textit{elaboration expressions} from the \(z_l\) in \(C\). For example, if we constructed the following function:

\[
\lambda f.\text{argmin}(\lambda x.\text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{ori}) \land f(x), \lambda y.\text{fare}(y))
\]

Another possible option is the expression \(\lambda x.!(\langle e,t\rangle \land \text{day}(x, \text{wed}) \land \text{depart}(x) > 1700)\). However, there is no obvious way to resolve the \(!!(e,t)\) 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:

\[
\lambda x. \text{flight}(x) \land \text{from}(x, \text{pit}) \land \text{to}(x, \text{la}) \\
\land \text{day}(x, \text{thur}) \land \text{during}(x, \text{afternoon})
\]

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

\[
\lambda x. !\langle e, t \rangle \land \text{day}(x, \text{thur}) \land \text{during}(x, \text{afternoon})
\]

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):

(3)

\[
\lambda x. \text{flight}(x) \land \text{from}(x, \text{pit}) \land \text{to}(x, \text{la}) \\
\land \text{day}(x, \text{thur}) \land \text{during}(x, \text{evening})
\]

Substituting this expression directly would produce the following logical form:

\[
\lambda x. \text{flight}(x) \land \text{from}(x, \text{pit}) \land \text{to}(x, \text{la}) \\
\land \text{day}(x, \text{thur}) \land \text{during}(x, \text{evening}) \\
\land \text{day}(x, \text{thur}) \land \text{during}(x, \text{afternoon})
\]

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:

\[
\lambda x. \text{flight}(x) \land \text{from}(x, \text{pit}) \land \text{to}(x, \text{la})
\]

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.
- \( R_{(e,t)}(C) \): 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

\[
\lambda x. \text{flight}(x) \land \text{from}(x, \text{pit}) \land \text{to}(x, \text{la})
\]

then \( R_e(C) = \{ \text{pit}, \text{la} \} \) and \( R_{(e,t)}(C) \) 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

\[
\lambda x. \text{flight}(x) \land \text{from}(x, \text{pit}) \land \text{to}(x, \text{la})
\]

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

**Derivations** We now formally define a *derivation* to be a sequence \( d = (l, s_1, \ldots, s_m) \). \( l \) is a CCG parse that constructs a context-independent logical form \( \pi \) with \( m - 1 \) reference expressions. 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 \((l, l', f, r_1, \ldots, r_p)\). This operator selects a reference \( f \) in the input logical form \( l \) and an appropriately typed expression \( r \) from either \( R_e(C) \) or \( R_{(e,t)}(C) \). 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 \( b_q(l) \).

\(^1\)In practice, \( \pi \) 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 \( R_e(C), R_{(e,t)}(C) \), 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 \( R_e(z) = \bigcup_{i=1}^{j-1} R_e(z_i) \).

For example, if \( z \) is \( \arg\min(\lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}), \lambda y. \text{fare}(y)) \), the resulting set is \( R_e(z) = \{\text{mil}, \text{orl}, z\} \), where \( z \) is included because the entire \( \arg\min \) expression has type \( e \).

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

The set \( R_{(e,t)}(z) \) contains all of the \( (e,t) \)-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:

\[ \lambda y. \exists x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \text{phi}) \land \text{during}(x, \text{morning}) \land \text{aircraft}(x) = y \]

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

\[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \text{phi}) \land \text{during}(x, \text{morning}) \]

constructed from the variable \( x \), where the subexpression \( \text{aircraft}(x) = y \) 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 (conjunctive or disjunctive) 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 \( \theta \) 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.

\(^7\)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 \( \{I_i: i = 1 \ldots n\} \). Each \( I_i \) is a sequence \( \{w_{i,j}, z_{i,j}: j = 1 \ldots n_i\} \) where \( w_{i,j} \) is a sentence and \( z_{i,j} \) is a logical form. Number of training iterations \( T \). Initial parameters \( \theta \).

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 \( t = 1 \ldots T, i = 1 \ldots n_i \) (Iterate interactions)
- Reset \( C \) = \{\}. (Reset context)
- For \( j = 1 \ldots n_i \) (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 \( \theta \).

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 \( R_c(z_i) \), \( R_{(c,t)}(z_i) \), 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; Senieff, 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,
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.

<table>
<thead>
<tr>
<th>Interactions</th>
<th>Train</th>
<th>Dev.</th>
<th>Test</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentences</td>
<td>2099</td>
<td>99</td>
<td>127</td>
<td>526</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>System</th>
<th>Partial Match</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Rec.</td>
</tr>
<tr>
<td>Full Method</td>
<td>95.0</td>
<td>96.5</td>
</tr>
<tr>
<td>Miller et al.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Performance on the ATIS DEC94 test set.

<table>
<thead>
<tr>
<th>Limited Context</th>
<th>Partial Match</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Rec.</td>
</tr>
<tr>
<td>$M = 0$</td>
<td>96.2</td>
<td>57.3</td>
</tr>
<tr>
<td>$M = 1$</td>
<td>94.9</td>
<td>91.6</td>
</tr>
<tr>
<td>$M = 2$</td>
<td>94.8</td>
<td>93.2</td>
</tr>
<tr>
<td>$M = 3$</td>
<td>94.5</td>
<td>94.3</td>
</tr>
<tr>
<td>$M = 4$</td>
<td>94.9</td>
<td>92.9</td>
</tr>
<tr>
<td>$M = 10$</td>
<td>94.2</td>
<td>94.0</td>
</tr>
</tbody>
</table>

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


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


