LEARNING COMPOSITIONALITY

Ignacio Cases with Christopher Potts
Stanford University
Meaning and Semantic Representation

**Compositional Semantics**

- **Message**
  - Utterances in natural language

- **Semantic Parsing**
  - Query Languages (SQL)
  - Lambda calculi
  - Programming languages (*Scala*)

- **Semantic Representation**

- **Execution**
  - Denotation
    - Model
    - DB
    - Mental Status
    - World
Meaning and Semantic Representation
Compositional Semantics

Message: one plus two

Semantic Representation: Add(1, 2)

Execution: Add(Neg(2), 3)

Denotation: crane

Message: minus two plus three

Semantic Representation: Add(Neg(2), 3)

Execution: Neg(Add(2, 3))

Denotation: crane (Noun(crane, bird), Noun(crane, machine))

Message: crane

Semantic Representation: Noun(crane, bird), Noun(crane, machine)
Meaning and Semantic Representation

Compositional Semantics

Message $\xrightarrow{\text{Semantic Parsing}}$ Semantic Representation $\xrightarrow{\text{Execution}}$ Denotation

two is less than three

LessThan(2,3) T

the sum of one to four

foldRight(zero)(Add(_,_)) 10
“The meaning of a sentence is a function of the meanings of the parts and of the way they are syntactically combined.”

Partee (1995)
Compositionality Principle

Compositional Semantics

old linguists and engineers

minus two plus three
Compositionality Principle

Compositional Semantics

old linguists and engineers

minus two plus three
Compositionality Principle

Compositional Semantics

(\text{old linguists}) \text{ and } \text{engineers}

(\text{minus two}) \text{ plus } \text{three}
Some trees:

```
NP
  E → T
  AP(E → T) → E → T
  A(E → T) → E → T
  old
    NP
      E → T
      N
      E → T
    linguists
    NP
      E → T
      N
      E → T
    engineers
```

```
NP
  E → T
  AP(E → T) → E → T
  A(E → T) → E → T
  minus
    NP
      E → T
      N
      E → T
    two
    NP
      E → T
      N
      E → T
    plus
      NP
        E → T
        N
        E → T
      three
```

**old** (linguists and engineers)  
**minus** (two plus three)
“compositionality characterizes the recursive nature of the linguistic ability required to generalize to a creative capacity, and learning details the conditions under which such an ability can be acquired from data.”

Liang and Potts (2015: 356)
Learning Compositionality

Percy Liang
University of California, Berkeley
Michael I. Jordan
University of California, Berkeley
Dan Klein
University of California, Berkeley

Suppose we want to build a system that answers a natural language question by representing its semantics as a logical form and computing the answer given a structured database of facts. The core part of such a system is the semantic parser that maps questions to logical forms. Semantic parsers are typically trained from examples of questions annotated with their target logical forms, but this type of annotation is expensive.

Our goal is to instead learn a semantic parser from question–answer pairs, where the logical form is modeled as a latent variable. We develop a new semantic formalism, dependency-based compositional semantics (DCS) and define a log-linear distribution over DCS logical forms. The model parameters are estimated using a simple procedure that alternates between beam search and numerical optimization. On two standard semantic parsing benchmarks, we show that our system obtains comparable accuracy to state-of-the-art systems that do require annotated logical forms.

1. Introduction

One of the major challenges in natural language processing (NLP) is building systems that both handle complex linguistic phenomena and require minimal human effort. The difficulty of achieving both criteria is particularly evident in training semantic parsers, where annotating linguistic expressions with their associated logical forms is expensive but until recently, seemingly unavoidable. Advances in learning latent-variable models, however, have made it possible to progressively reduce the amount of supervision.

No rights reserved. This work was authored as part of the Contributor's official duties as an Employee of the United States Government and is therefore a work of the United States Government. No copyright is available in the United States.

Learning Dependency-Based Compositional Semantics

Percy Liang

∗ Computer Science Division, University of California, Berkeley, CA 94720, USA. E-mail: pliang@cs.stanford.edu.

∗∗ Computer Science Division and Department of Statistics, University of California, Berkeley, CA 94720, USA. E-mail: jordan@cs.berkeley.edu.

† Computer Science Division, University of California, Berkeley, CA 94720, USA. E-mail: klein@cs.berkeley.edu.

Submission received: 12 September 2011; revised submission received: 19 February 2012; accepted for publication: 18 April 2012.

doi:10.1162/COLI00127
Learning Challenge
Learning Compositionality

Figure 2
Our statistical methodology consists of two steps: (i) semantic parsing ($z \sim p(z \mid x; \theta)$): an utterance $x$ is mapped to a logical form $z$ by drawing from a log-linear distribution parametrized by a vector $\theta$; and (ii) evaluation ($[z]_w$): the logical form $z$ is evaluated with respect to the world $w$ (database of facts) to deterministically produce an answer $y$. The figure also shows an example configuration of the variables around the graphical model. Logical forms $z$ are represented as labeled trees. During learning, we are given $w$ and ($x$, $y$) pairs (shaded nodes) and try to infer the latent logical forms $z$ and parameters $\theta$.

Although liberating ourselves from annotated logical forms reduces cost, it does increase the difficulty of the learning problem. The core challenge here is program induction: One a example ($x$, $y$), we need to efficiently search over the exponential space of possible logical forms ($z$) and find ones that produce the target answer $y$, a computationally daunting task. There is also a statistical challenge: How do we parametrize the mapping from utterance $x$ to logical form $z$ so that it can be learned from only the indirect signal $y$? To address these challenges, we must first discuss the issue of semantic representation. There are several questions here: (i) what

Liang et al. (2013: 391)
Structured Prediction

Learning Compositionality

Four pieces

- initial grammar (refined during the learning)
### Structured Prediction

#### Learning Compositionality

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Add</th>
<th>Sub</th>
<th>Mul</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>

**Pages:**

- **Page 5:**
  - Add: 1
  - Sub: 2
  - Mul: ...

- **Page 6:**
  - Add: 1
  - Sub: 2
  - Mul: ...

- **Page 7:**
  - Add: 1
  - Sub: 2
  - Mul: ...
Structured Prediction
Learning Compositionality

Add 4 1 5
Add 4 2 6
Add 6 1 7
Structured Prediction

Learning Compositionality

Four pieces

- initial grammar (refined during the learning)
- a feature representation of the data
- an objective function
- an algorithm for optimizing the objective function

Liang and Potts (2015)
lazy val expr: Parser[Expr] = {
  expr ~ "plus" ~ expr ^^ { (e1, _, e2) => Add(e1, e2, "plus") },
  expr ~ "plus" ~ expr ^^ { (e1, _, e2) => Mul(e1, e2, "plus") },
  expr ~ "plus" ~ expr ^^ { (e1, _, e2) => Sub(e1, e2, "plus") },
  ...
  | term
}

val numbers = List("one", "two", "three", ..., "nine")

val terms = for {
  n <- numbers
  x <- 1 to 9
} yield (n ^^ { _ => IntLit(x, n) })

val term = terms.reduceLeft(_ | _)
Algebraic Data Types
Synthesis Framework

sealed trait Expr

trait BinExpr extends Expr
trait UnExpr extends Expr

case class Add(e1: Expr, e2: Expr, label: String) extends BinExpr
case class Sub(e1: Expr, e2: Expr, label: String) extends BinExpr
case class Mul(e1: Expr, e2: Expr, label: String) extends BinExpr
case class Neg(e: Expr, label: String) extends UnExpr

case class IntLit(i: Int, label: String) extends Expr
Denotations as Catamorphisms
Synthesis Framework

“compositionality outlines a recursive interpretation process in which the lexical items are listed as base cases and the recursive clauses define the modes of combination.”

Liang and Potts (2015: 359)

“Fold is a generic transform for any algebraic data type”

Noel Welsh (2015)

Fold provides the appropriate abstraction over structural recursion
def foldt[A,B](f: Expr => Int, c: Int, g: Expr => (Int => Int => Int))(t: Expr): Int =
  t match {
    case IntLit(a, l) => f(IntLit(a, l))
    case u: UnExpr => g(u)(c)(foldt(f, c, g)(u.e))
    case b: BinExpr => g(b)(foldt(f, c, g)(b.e1))(foldt(f, c, g)(b.e2))
  }
Algebraic Data Types
Synthesis Framework

minus two plus three

...
def evaluate[U,S,D](phi: Phi[U,S,D], optimizer: Optimizer[U,S,D],
train: Data[U,S,D], test: Data[U,S,D], classes: U => List[S],
numIter: Int, eta: Double, transform: S => D, cost: (S, S) => Double) = {

  val w = optimizer(train, phi, numIter, eta, classes, transform, cost)

  val featureWeights = for((f, value) <- w.toSeq.sortBy(_._2)) yield (f, value)

  ...

  }
Polymorphic implementation of SGD

```python
def latentSGD[U,S,D](dataset: List[(U, S, D)], phi: Phi[U,S,D], numIter: Int,
eta: Double, classes: U => List[S], transform: S => D,
cost: (S, S) => Double): Weight[U,D] = {
  ...
  for (t <- 0 until numIter) {
    for ((x, ys, d) <- shuffledData) {
      val monadicClass = (z: U) => for {
        zd <- classes(z)
        if transform(zd) == d
      } yield zd
      // Get the best viable candidate given the current weights:
      val y = predict(x, phi, w, monadicClass, transform)
      // Get all (score, y') pairs:
      val scores = for(yAlt <- classes(x)) yield (score(x, yAlt, phi, w) + cost(y, yAlt), yAlt)
      // Get the maximal score
      // Get all the candidates with the max score and chose one randomly
      // Update the weights
      ...
    }
  }
}
```

Synthesis Framework

Results

Interpretive

Feature function: Catamorphism

Learned feature weights

Training:

Input: one plus one
Gold: $1 + 1$
Prediction: $1 + 1$
Correct: true

Input: one plus two
Gold: $1 + 2$
Prediction: $1 + 2$
Correct: true

Input: one plus three
Gold: $1 + 3$
Prediction: $1 + 3$
Correct: true
Results
Synthesis Framework

Test:

Input: minus three
Gold: -3
Prediction: -3
Correct: true

Input: three plus two
Gold: 3 + 2
Prediction: 3 + 2
Correct: true

Input: minus four
Gold: -4
Prediction: -1
Correct: false
Future directions

Synthesis Framework

Pruning trees applying a type checker

Tractability: avoid combinatoric explosion

Scale up

Deep Learning