

Lecture 2: Mixing Compositional and Statistical Semantics

Kyle Richardson

kyle@ims.uni-stuttgart.de

May 20, 2016

Plan

- ▶ **main paper:** Liang and Potts 2015 (conceptual basis of class)
- ▶ **secondary:** Mooney 2007 (semantic parsing big ideas),
Domingos 2012 (remarks about ML)

Classical Semantics vs. Statistical Semantics (caricature)

- ▶ **Logical Semantics:** Logic, algebra, set theory
 - ▶ **compositional analysis**, **beyond words**, inference, brittle.
- ▶ **Statistical Semantics:** Optimization, algorithms, geometry
 - ▶ **distributional analysis**, **word-based**, grounded, shallow.

“The two types of approaches share the long-term vision of achieving deep natural language understanding...”

Montague-style Compositional Semantics

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

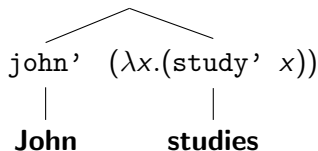
Montague-style Compositional Semantics

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

$$(\lambda x.(\text{study}' x))(\text{john}) \rightarrow (\text{study}' \text{john}') \rightarrow \{True, False\}$$



A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
```

A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> fun_application(study, "bill") ## What will we get?
```

A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> fun_application(study, "bill") ## What will we get?
>>> False
```


A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> fun_application(study, "mary") ## What will we get?
```

A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *John studies.*

$$\begin{aligned}\text{john}' &\longrightarrow \text{"John"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"studies"}\end{aligned}$$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> fun_application(study, "mary") ## What will we get?
>>> True
```

Montague-style Compositional Semantics

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$\text{bill}' \longrightarrow \text{"Bill"}$

$(\lambda x.(\text{study}' x)) \longrightarrow \text{"study"}$

$(\lambda f.\lambda x.(\text{not } (f x))) \longrightarrow \text{"does not"}$

Montague-style Compositional Semantics

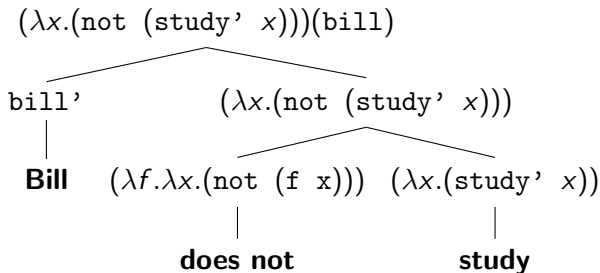
Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$\text{bill}' \rightarrow \text{"Bill"}$

$(\lambda x.(\text{study}' x)) \rightarrow \text{"study"}$

$(\lambda f.\lambda x.(\text{not } (f x))) \rightarrow \text{"does not"}$



A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

`bill'` \longrightarrow **"Bill"**

`(λx .(study' x))` \longrightarrow **"study"**

`(λf . λx .(not (f x)))` \longrightarrow **"does not"**

```
>>> students_studying = set(["john", "mary"])
```

```
>>> study = lambda x : x in students_studying
```

```
>>> fun_application = lambda fun, val : fun(val)
```

```
>>> neg = lambda F : (lambda x : not F(x))
```

A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$\text{bill}' \longrightarrow \text{"Bill"}$
 $(\lambda x.(\text{study}' x)) \longrightarrow \text{"study"}$
 $(\lambda f.\lambda x.(\text{not } (f x))) \longrightarrow \text{"does not"}$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> neg = lambda F : (lambda x : not F(x))
>>> neg(study)("bill") # True
```

A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$\text{bill}' \longrightarrow \text{"Bill"}$
 $(\lambda x.(\text{study}' x)) \longrightarrow \text{"study"}$
 $(\lambda f.\lambda x.(\text{not } (f x))) \longrightarrow \text{"does not"}$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> neg = lambda F : (lambda x : not F(x))
>>> neg(study)("bill") # True
>>> fun_application(neg,study)("bill")
```

A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$\text{bill}' \longrightarrow \text{"Bill"}$
 $(\lambda x.(\text{study}' x)) \longrightarrow \text{"study"}$
 $(\lambda f.\lambda x.(\text{not } (f x))) \longrightarrow \text{"does not"}$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> neg = lambda F : (lambda x : not F(x))
>>> neg(study)("bill") # True
>>> fun_application(neg,study)("bill")
>>> fun_application(fun_application(neg,study),"bill")
```


A mini functional interpreter (python)

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$\text{bill}' \longrightarrow \text{"Bill"}$
 $(\lambda x.(\text{study}' x)) \longrightarrow \text{"study"}$
 $(\lambda f.\lambda x.(\text{not } (f x))) \longrightarrow \text{"does not"}$

```
>>> students_studying = set(["john", "mary"])
>>> study = lambda x : x in students_studying
>>> fun_application = lambda fun, val : fun(val)
>>> neg = lambda F : (lambda x : not F(x))
>>> neg(study)("bill") # True
>>> fun_application(neg,study)("bill")
>>> fun_application(fun_application(neg,study),"bill")
>>> neg(neg(sleep))("bill")
```

Montague-style Compositional Semantics: What's needed

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$$\begin{aligned}\text{bill}' &\longrightarrow \text{"Bill"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"study"} \\ (\lambda f.\lambda x.(\text{not } (f x))) &\longrightarrow \text{"does not"}\end{aligned}$$

- ▶ Grammar rules for building syntactic structure.
- ▶ Interpretation rules to composing meaning.
- ▶ **Decoding algorithm for generating structures**

Montague-style Compositional Semantics: Issues

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$$\begin{aligned}\text{bill}' &\longrightarrow \text{"Bill"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"study"} \\ (\lambda f.\lambda x.(\text{not } (f x))) &\longrightarrow \text{"does not"}\end{aligned}$$

Features and (Computational) Issues:

- ▶ compositional, provides a full analysis.
- ▶ supports further inferencing

Montague-style Compositional Semantics: Issues

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$$\begin{aligned}\text{bill}' &\longrightarrow \text{"Bill"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"study"} \\ (\lambda f.\lambda x.(\text{not } (f x))) &\longrightarrow \text{"does not"}\end{aligned}$$

Features and (Computational) Issues:

- ▶ compositional, provides a full analysis.
- ▶ supports further inferencing
- ▶ **issue:** Does not provide an analysis of words (**not grounded**).

Montague-style Compositional Semantics: Issues

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$$\begin{aligned}\text{bill}' &\longrightarrow \text{"Bill"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"study"} \\ (\lambda f.\lambda x.(\text{not } (f x))) &\longrightarrow \text{"does not"}\end{aligned}$$

Features and (Computational) Issues:

- ▶ compositional, provides a full analysis.
- ▶ supports further inferencing
- ▶ **issue:** Does not provide an analysis of words (**not grounded**).
- ▶ **issue:** Is brittle, cannot handle uncertainty.

Montague-style Compositional Semantics: Issues

Principle of Compositionality: The meaning of a complex expression is a function of the meaning of its parts and the rules that combine them.

Example: *Bill does not study.*

$$\begin{aligned}\text{bill}' &\longrightarrow \text{"Bill"} \\ (\lambda x.(\text{study}' x)) &\longrightarrow \text{"study"} \\ (\lambda f.\lambda x.(\text{not } (f x))) &\longrightarrow \text{"does not"}\end{aligned}$$

Features and (Computational) Issues:

- ▶ compositional, provides a full analysis.
- ▶ supports further inferencing
- ▶ **issue:** Does not provide an analysis of words (**not grounded**).
- ▶ **issue:** Is brittle, cannot handle uncertainty.
- ▶ **issue:** Says nothing about how the translation to logic works.

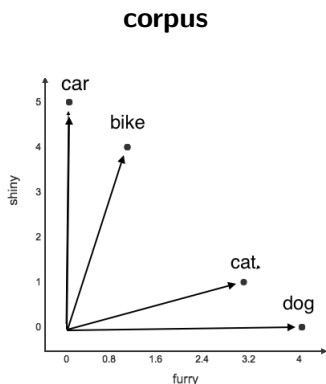
Statistical Approaches to Semantics

Statistical semantics hypothesis: “Statistical patterns of human word usage can be used to figure out what people mean” Turney et al. (2010)

corpus		word-context matrix			
The furry dog is walking outside...		furry	walking	shiny	driving
The shiny car is driving ...	<i>dog</i>	10	20	0	0
A furry cat is walking around...	<i>cat</i>	12	25	2	0
A shiny bike is driving	<i>car</i>	0	0	23	26
....	<i>bike</i>	0	1	30	25

Statistical Approaches to Semantics

Statistical semantics hypothesis: “Statistical patterns of human word usage can be used to figure out what people mean” Turney et al. (2010)



word-context matrix

	furry	walking	shiny	driving
<i>dog</i>	4	20	0	0
<i>cat</i>	3	25	2	0
<i>car</i>	0	0	5	26
<i>bike</i>	1	1	4	25

Example Tasks and Applications: Turney et al. (2010)

Statistical semantic models are often used in downstream classification or clustering tasks/applications.

- ▶ **Term-document matrices**

- ▶ Document retrieval/clustering/classification.
- ▶ Question Answering and Retrieval.
- ▶ Essay scoring.

- ▶ **Word-Context Matrices**

- ▶ Word similarity/clustering/classification
- ▶ Word-sense disambiguation
- ▶ Automatic thesaurus generation/paraphrasing

- ▶ **Pair-pair matrices**

- ▶ Relational similarity/clustering/classification.
- ▶ Analogy comparison.

Statistical Approaches to Semantics

Statistical semantics hypothesis: “Statistical patterns of human word usage can be used to figure out what people mean” Turney et al. (2010)

corpus		word-context matrix			
The furry dog is walking outside...		furry	walking	shiny	driving
The shiny car is driving ...	<i>dog</i>	10	20	0	0
A furry cat is walking around...	<i>cat</i>	12	25	2	0
A shiny bike is driving	<i>car</i>	0	0	23	26
....	<i>bike</i>	0	1	30	25

Features and Issues (caricature):

- ▶ Robust, requires little manual effort, **grounded**
- ▶ Can provide rich analysis of content words.

Statistical Approaches to Semantics

Statistical semantics hypothesis: “Statistical patterns of human word usage can be used to figure out what people mean” Turney et al. (2010)

corpus		word-context matrix			
The furry dog is walking outside...		furry	walking	shiny	driving
The shiny car is driving ...	<i>dog</i>	10	20	0	0
A furry cat is walking around...	<i>cat</i>	12	25	2	0
A shiny bike is driving	<i>car</i>	0	0	23	26
....	<i>bike</i>	0	1	30	25

Features and Issues (caricature):

- ▶ Robust, requires little manual effort, **grounded**
- ▶ Can provide rich analysis of content words.
- ▶ **issue:** Hard to scale beyond words.

Statistical Approaches to Semantics

Statistical semantics hypothesis: “Statistical patterns of human word usage can be used to figure out what people mean” Turney et al. (2010)

corpus		word-context matrix			
The furry dog is walking outside...		furry	walking	shiny	driving
The shiny car is driving ...	<i>dog</i>	10	20	0	0
A furry cat is walking around...	<i>cat</i>	12	25	2	0
A shiny bike is driving	<i>car</i>	0	0	23	26
....	<i>bike</i>	0	1	30	25

Features and Issues (caricature):

- ▶ Robust, requires little manual effort, **grounded**
- ▶ Can provide rich analysis of content words.
- ▶ **issue:** Hard to scale beyond words.
- ▶ **issue:** In general, hard to model logical operations, shallow.

Mixing compositional and statistical semantics

Desiderata: Want a model of semantics that is robust, reflects real-word usage and learnable, but one that is also compositional.

Mixing compositional and statistical semantics

Desiderata: Want a model of semantics that is robust, reflects real-word usage and learnable, but one that is also compositional.

- ▶ **Generalization**

Mixing compositional and statistical semantics

Desiderata: Want a model of semantics that is robust, reflects real-word usage and learnable, but one that is also compositional.

- ▶ **Generalization**

- ▶ **Logical semantics:** generalize using composition and abstract recursive structures.

Mixing compositional and statistical semantics

Desiderata: Want a model of semantics that is robust, reflects real-word usage and learnable, but one that is also compositional.

- ▶ **Generalization**

- ▶ **Logical semantics:** generalize using composition and abstract recursive structures.
- ▶ **Machine Learning (classification):** learns generalizations through real-world examples (e.g. target input-output)

Mixing compositional and statistical semantics

Desiderata: Want a model of semantics that is robust, reflects real-word usage and learnable, but one that is also compositional.

- ▶ **Generalization**

- ▶ **Logical semantics:** generalize using composition and abstract recursive structures.
 - ▶ **Machine Learning (classification):** learns generalizations through real-world examples (e.g. target input-output)
- ▶ **Bridge:** get our learning to target compositional structures.

A simple model: Liang and Potts

Model: a simple discriminative learning framework.

- ▶ **compositional model:** (semantic) context-free grammar.
- ▶ **learning model:** linear classification and first-order optimization.

Compositional Model:

Linguistic Objects: $\langle u, s, d \rangle$

- ▶ **u:** utterance
- ▶ **s:** semantic representation (symbolized as \hat{u})
- ▶ **d:** denotation (symbolized as $\llbracket s \rrbracket$)

Compositional Model:

Linguistic Objects: $\langle u, s, d \rangle$

- ▶ **u:** utterance
- ▶ **s:** semantic representation (symbolized as \hat{u})
- ▶ **d:** denotation (symbolized as $\llbracket s \rrbracket$)

Example: $\langle \text{'seven minus five'}, (-7\ 5), 2 \rangle$

Compositional Model:

Linguistic Objects: $\langle u, s, d \rangle$

- ▶ **u:** utterance
- ▶ **s:** semantic representation (symbolized as \hat{u})
- ▶ **d:** denotation (symbolized as $\llbracket s \rrbracket$)

Example: $\langle \text{'seven minus five'}, (-\ 7\ 5), 2 \rangle$
 $\langle \text{'two minus two times two'}, (*\ (-\ 2\ 2)\ 2), 0 \rangle$

Compositional Model:

Linguistic Objects: $\langle u, s, d \rangle$

- ▶ **u:** utterance
- ▶ **s:** semantic representation (symbolized as \hat{u})
- ▶ **d:** denotation (symbolized as $\llbracket s \rrbracket$)

Example: $\langle \text{'seven minus five'}, (-\ 7\ 5), 2 \rangle$
 $\langle \text{'two minus two times two'}, (*\ (-\ 2\ 2)\ 2), 0 \rangle$

semantic parsing: $u \rightarrow s$

Compositional Model:

Linguistic Objects: $\langle u, s, d \rangle$

- ▶ **u:** utterance
- ▶ **s:** semantic representation (symbolized as \hat{u})
- ▶ **d:** denotation (symbolized as $\llbracket s \rrbracket$)

Example: $\langle \text{'seven minus five'}, (-\ 7\ 5), 2 \rangle$

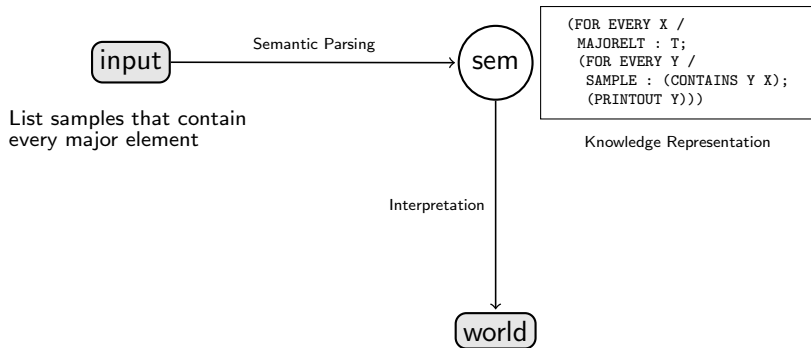
$\langle \text{'two minus two times two'}, (*\ (-\ 2\ 2)\ 2), 0 \rangle$

semantic parsing: $u \rightarrow s$

interpretation: $s \rightarrow d$

Computational Modeling: The full picture

► Standard processing pipeline



$$\llbracket sem \rrbracket = \{S10019, S10059, \dots\}$$

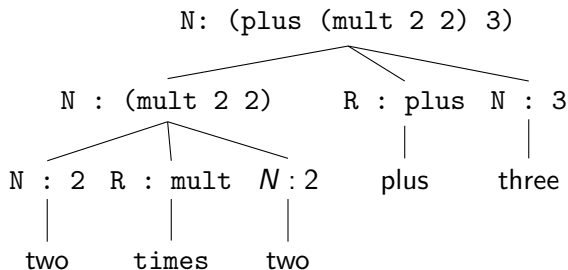
Compositional Model: Context-free grammar

- provides the background grammar and interpretation rules

Syntax	Semantic representation	Denotation
$N \rightarrow one$	1	1
$N \rightarrow two$	2	2
\vdots	\vdots	\vdots
$R \rightarrow plus$	+	the R such that $R(x, y) = x + y$
$R \rightarrow minus$	-	the R such that $R(x, y) = x - y$
$R \rightarrow times$	\times	the R such that $R(x, y) = x * y$
$S \rightarrow minus$	\neg	the f such that $f(x) = -x$
$N \rightarrow S N$	$\ulcorner S \urcorner \ulcorner N \urcorner$	$[\ulcorner S \urcorner](\llbracket \ulcorner N \urcorner \rrbracket)$
$N \rightarrow N_L R N_R$	$(\ulcorner R \urcorner \ulcorner N_L \urcorner \ulcorner N_R \urcorner)$	$[\ulcorner R \urcorner](\llbracket \ulcorner N_L \urcorner \rrbracket \llbracket \ulcorner N_R \urcorner \rrbracket)$

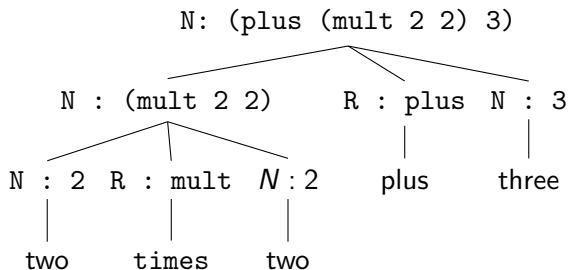
Compositional Model: Context-free grammar

- ▶ provides the background grammar and interpretation rules
- ▶ **example:** $u = \text{two times two plus three}$



Compositional Model: Context-free grammar

- ▶ provides the background grammar and interpretation rules
- ▶ **example:** $u = \text{two times two plus three}$

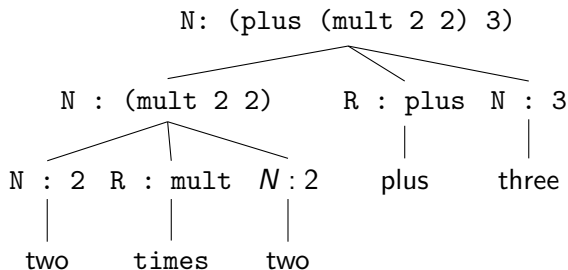


>>> plus = lambda x,y : x + y

>>> mult = lambda x,y : x * y

Compositional Model: Context-free grammar

- ▶ provides the background grammar and interpretation rules
- ▶ **example:** $u = \text{two times two plus three}$



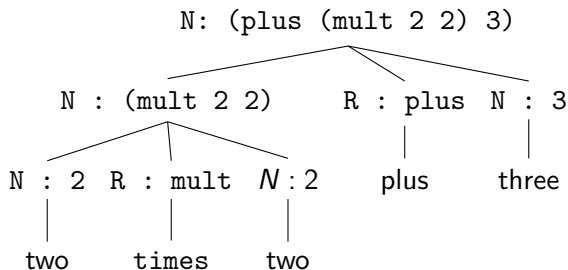
>>> plus = lambda x,y : x + y

>>> mult = lambda x,y : x * y

>>> plus(2,2) # 4

Compositional Model: Context-free grammar

- ▶ provides the background grammar and interpretation rules
- ▶ **example:** $u = \text{two times two plus three}$



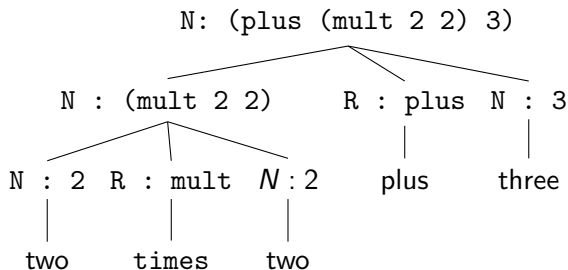
>>> plus = lambda x,y : x + y

>>> mult = lambda x,y : x * y

>>> plus(plus(2,3),2) # 7

Compositional Model: Context-free grammar

- ▶ provides the background grammar and interpretation rules
- ▶ **example:** $u = \text{two times two plus three}$



>>> plus = lambda x,y : x + y

>>> mult = lambda x,y : x * y

>>> plus(mult(2,2),3) # 7

Compositional Model: Components

- ▶ **Components:**

- ▶ Grammar rules for building syntactic structure. ✓

Compositional Model: Components

► **Components:**

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓

Compositional Model: Components

► Components:

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓
- **Decoding algorithm for generating structures** × (later lecture)

Compositional Model: Components

► Components:

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓
- **Decoding algorithm for generating structures** × (later lecture)
- **Rule extraction** × (later lecture)

Compositional Model: Components

► Components:

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓
- **Decoding algorithm for generating structures** × (later lecture)
- **Rule extraction** × (later lecture)

► Issues:

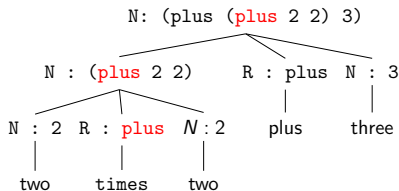
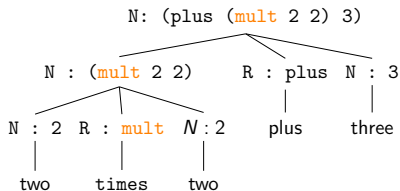
Compositional Model: Components

► Components:

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓
- **Decoding algorithm for generating structures** × (later lecture)
- **Rule extraction** × (later lecture)

► Issues:

- **example:** $u = \text{two times two plus three}$



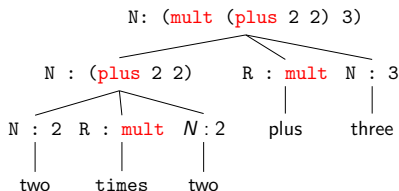
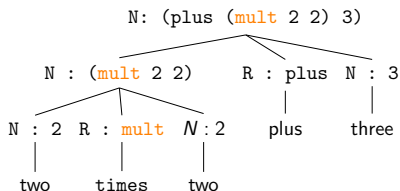
Compositional Model: Components

► Components:

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓
- **Decoding algorithm for generating structures** × (later lecture)

► Issues:

- **example:** $u = \text{two times two plus three}$



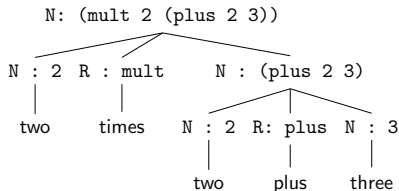
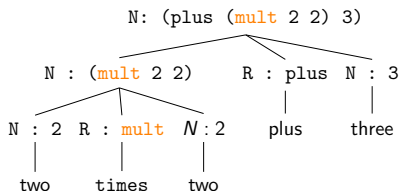
Compositional Model: Components

► Components:

- Grammar rules for building syntactic structure. ✓
- Interpretation rules to composing meaning. ✓
- **Decoding algorithm for generating structures** × (later lecture)

► Issues:

- **example:** $u = \text{two times two plus three}$



Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).

Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).

Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).
- ▶ Components

Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).
- ▶ Components
 - ▶ training data $D = \{(x_i, y_i) | i \dots n\}$

Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).
- ▶ Components
 - ▶ training data $D = \{(x_i, y_i) | i \dots n\}$
 - ▶ feature representation of data

Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).
- ▶ Components
 - ▶ training data $D = \{(x_i, y_i) | i \dots n\}$
 - ▶ feature representation of data
 - ▶ scoring and objective function

Learning Model

- ▶ **Goal:** Helps us learn the correct derivations and handle uncertainty (word mappings, composition).
- ▶ **Classifier:** “a system that inputs a vector of discrete and/or continuous **feature values** and outputs a single discrete value, the **class**.” Domingos (2012).
- ▶ Components
 - ▶ training data $D = \{(x_i, y_i) | i \dots n\}$
 - ▶ feature representation of data
 - ▶ scoring and objective function
 - ▶ optimization procedure

Training data

Goal: Find the **correct** derivations and output using our compositional model

Training data

Goal: Find the **correct** derivations and output using our compositional model

Logical forms (more information)

- ▶ ($u = \text{'two minus two times two'}$, $s = (* (- 2 2) 2)$)

Training data

Goal: Find the **correct** derivations and output using our compositional model

Logical forms (more information)

- ▶ $(u = \text{'two minus two times two'}, s = (* (- 2 2) 2))$

Denotations (less information)

- ▶ $(u = \text{'two minus two times two'}, r = 0)$

Training data

Goal: Find the **correct** derivations and output using our compositional model

Logical forms (more information)

- ▶ $(u = \text{'two minus two times two'}, s = (* (- 2 2) 2))$

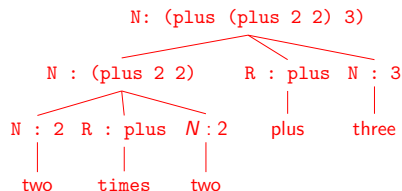
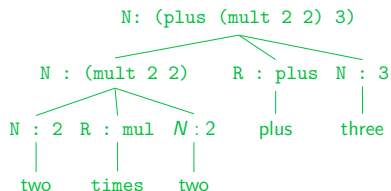
Denotations (less information)

- ▶ $(u = \text{'two minus two times two'}, r = 0)$

Weakly Supervised: In both cases, details are still hidden from the learner.

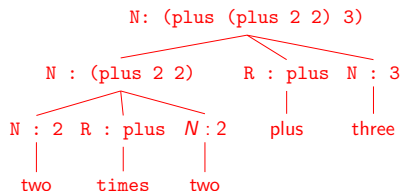
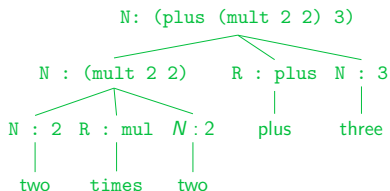
Learning from Semantic Representations

- **example:** (*two times two plus three*, (plus (mult 2 2) 3))



Learning from Semantic Representations

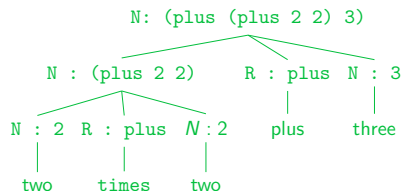
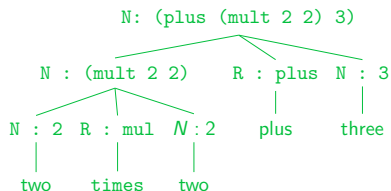
- **example:** (*two times two plus three*, (plus (mult 2 2) 3))



- **Trade off:** More information (good) but more annotation (bad)

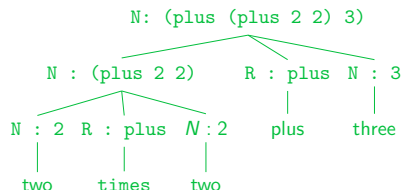
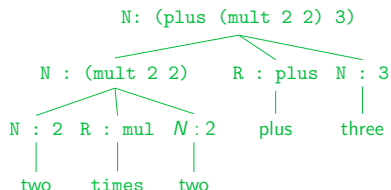
Learning from Denotations

- **example:** (*two times two plus three*, 7)



Learning from Denotations

- **example:** (*two times two plus three*, 7)



- **Trade off:** Less annotation (good) but less information (maybe bad)

Weak Supervision

Goal: Find the **correct** derivations and output using our compositional model

Logical forms (more information)

- ▶ ($u = \text{'two minus two times two'}$, $s = (* (- 2 2) 2)$)

Denotations (less information)

- ▶ ($u = \text{'two minus two times two'}$, $r = 0$)

“Current learning methods for NLP require annotating large corpora with supervisory information ...[e.g. pos tags, syntactic parse trees, semantic role labels] ... Building such corpora is an expensive, arduous task. **As one moves towards deeper semantic analysis the annotation task becomes increasingly more difficult and complex.**”

Mooney (2008)

Feature Representations: General Remark

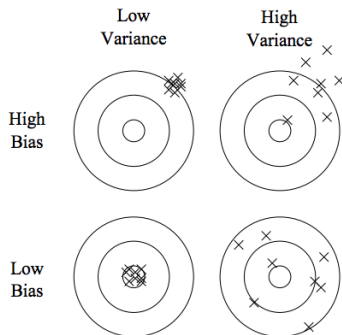
“At the end of the day, some machine learning projects succeed and fail. What makes the difference? Easily the most important factor is the features used.”

Domingos (2012)

	(x, y)	Feature representations $\phi(x, y)$		
		‘empty string’	‘last word’	‘all words’
Train	(twenty-five, O)	ϵ	five	[twenty, five]
	(thirty-one, O)	ϵ	one	[thirty, one]
	(forty-nine, O)	ϵ	nine	[forty, nine]
	(fifty-two, E)	ϵ	two	[fifty, two]
	(eighty-two, E)	ϵ	two	[eighty, two]
	(eighty-four, E)	ϵ	four	[eighty, four]
	(eighty-six, E)	ϵ	six	[eighty, six]
Test	(eighty-five, O)	$\epsilon \rightarrow E$	five $\rightarrow O$	[eighty, five] $\rightarrow E$

Feature selection and overfitting

"What if the knowledge and data we have are not sufficient to completely determine the correct classifier? Then we run the risk of just hallucinating a classifier (or parts of it) that is not grounded in reality .. This problem is called overfitting." Domingos (2012)



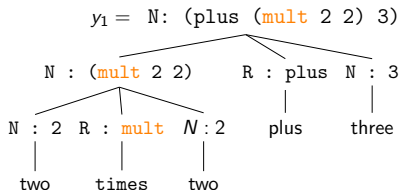
- ▶ **Bias:** Tendency to consistently learn the wrong thing.
- ▶ **Variance:** Tendency to learn random things irrespective of the real signal.

Good vs. Bad Feature Selection

	(x, y)	Feature representations $\phi(x, y)$		
		'empty string'	'last word'	'all words'
Train	(twenty-five, O)	ϵ	five	[twenty, five]
	(thirty-one, O)	ϵ	one	[thirty, one]
	(forty-nine, O)	ϵ	nine	[forty, nine]
	(fifty-two, E)	ϵ	two	[fifty, two]
	(eighty-two, E)	ϵ	two	[eighty, two]
	(eighty-four, E)	ϵ	four	[eighty, four]
	(eighty-six, E)	ϵ	six	[eighty, six]
Test	(eighty-five, O)	$\epsilon \rightarrow E$	five $\rightarrow O$	[eighty, five] $\rightarrow E$

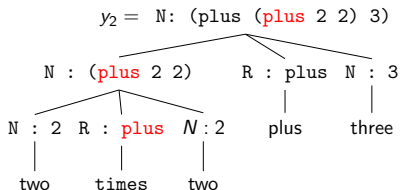
Feature Extraction Example

input: $x = \text{two times two plus three.}$



$\phi(x, y_1) =$

```
R : mult [ 'times' ] → 1
R : plus [ 'plus' ] → 1
top [ R : plus ] → 1
...
```



$\phi(x, y_2) =$

```
R : plus [ 'times' ] → 1
R : plus [ 'plus' ] → 1
top [ R : plus ] → 1
...
```

Scoring Function

(Linear) Score Function

- ▶ $\text{Score}_w(x, y) = w \cdot \phi(x, y) = \sum_{j=1}^d w_j \phi_j(x, y)$

Scoring Function

(Linear) Score Function

- ▶ $\text{Score}_w(x,y) = w \cdot \phi(x,y) = \sum_{j=1}^d w_j \phi(x,y)$
- ▶ weight vector $w = [w_1 = 0.1 \ w_2 = 0.2 \ w_3 = 0.0 \ \dots]$

Scoring Function

(Linear) Score Function

- ▶ $\text{Score}_w(x,y) = w \cdot \phi(x,y) = \sum_{j=1}^d w_j \phi(x,y)$
- ▶ weight vector $w = [w_1 = 0.1 \ w_2 = 0.2 \ w_3 = 0.0 \ \dots]$

$\phi(x,y_2) =$

$w_1 \text{ R : plus ['times'] } \rightarrow 1$
 $w_2 \text{ R : plus ['plus'] } \rightarrow 1$
 $w_3 \text{ top [R : plus] } \rightarrow 1$
...

$$\text{score}_w(x, y_2) = w \cdot \phi(x, y_2) = (0.1 * 1.0) + (0.2 * 1.0) + (0.0 * 1.0)$$

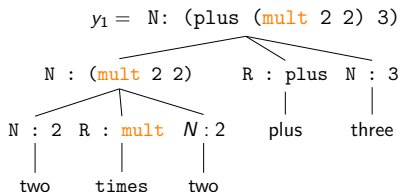
Scoring Function

(Linear) Score Function

- ▶ $\text{Score}_w(x, y) = w \cdot \phi(x, y) = \sum_{j=1}^d w_j \phi(x, y)$
- ▶ weight vector $w = [w_1 = 0.1 \ w_2 = 0.2 \ w_3 = 0.0 \ \dots]$
- ▶ prediction: $\arg\max_{y \in Y} \text{Score}_w(x, y)$

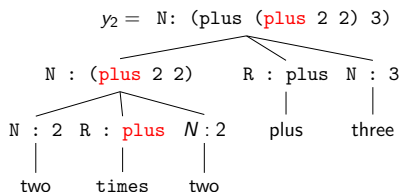
Objectives: What do we want to learn? (informal)

General Idea: want to learn a model (or weight vector) that can distinguish correct and incorrect derivations.



$\phi(x, y_1) =$

```
R : mult [ 'times' ] → 1
R : plus [ 'plus' ] → 1
top [ R : plus ] → 1
...
```

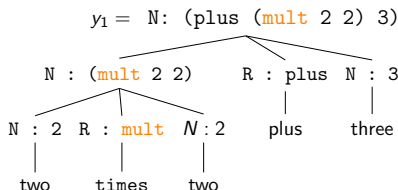


$\phi(x, y_2) =$

```
R : plus [ 'times' ] → 1
R : plus [ 'plus' ] → 1
top [ R : plus ] → 1
...
```

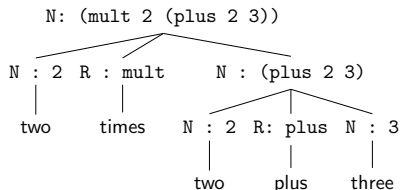
Objectives: What do we want to learn? (informal)

General Idea: want to learn a model (or weight vector) that can distinguish correct and incorrect derivations.



$\phi(x, y_1) =$

```
R : mult [ 'times' ] → 1
R : plus [ 'plus' ] → 1
plus [ R : mult ] → 1
...
```



$\phi(x, y_2) =$

```
R : plus [ 'times' ] → 1
R : plus [ 'plus' ] → 1
mult [ R : plus ] → 1
...
```


Objectives: What do we want to learn? (formal)

- ▶ **hinge loss:** (learning from logical forms)

$$\min_{w \in \mathbb{R}^d} \sum_{(x,y) \in D}^n \max_{y' \in Y} [Score_w(x, y') + c(y, y')] - Score_w(x, y)$$

- ▶ ('two minus two times two', $s = (* (- 2 2) 2)$)

Objectives: What do we want to learn? (formal)

- ▶ **hinge loss:** (learning from logical forms)

$$\min_{w \in \mathbb{R}^d} \sum_{(x,y) \in D}^n \max_{y' \in Y} [Score_w(x, y') + c(y, y')] - Score_w(x, y)$$

- ▶ ('two minus two times two', $s = (* (- 2 2) 2)$)
- ▶ **In English:** select parameters that minimize the cumulative loss over the training data.

Objectives: What do we want to learn? (formal)

- ▶ **hinge loss:** (learning from logical forms)

$$\min_{w \in \mathbb{R}^d} \sum_{(x,y) \in D}^n \max_{y' \in Y} [Score_w(x, y') + c(y, y')] - Score_w(x, y)$$

- ▶ ('two minus two times two', $s = (* (- 2 2) 2)$)
- ▶ **In English:** select parameters that minimize the cumulative loss over the training data.
- ▶ **Missing:** A decoding algorithm for generating Y (not trivial, Y might be very large).

Optimization: How do I achieve this objective?

- **Stochastic gradient descent:** An online learning and optimization algorithm (more about this in future lectures).

STOCHASTICGRADIENTDESCENT(\mathcal{D}, T, η)

\mathcal{D} : a set of training examples $(x, y) \in (\mathcal{X} \times \mathcal{Y})$

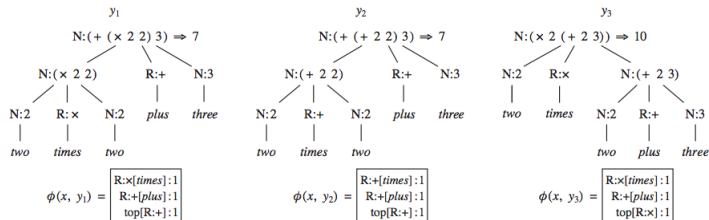
T : the number of passes to make through the data

$\eta > 0$: learning rate (e.g., $\frac{1}{\sqrt{T}}$)

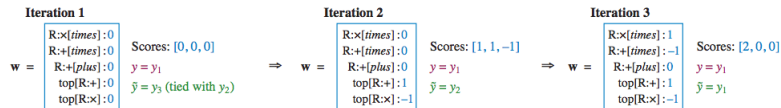
- 1 Initialize $\mathbf{w} \leftarrow \mathbf{0}$
- 2 Repeat T times
- 3 for each $(x, y) \in \mathcal{D}$ (in random order)
- 4 $\tilde{y} \leftarrow \arg \max_{y' \in \mathcal{Y}} \text{Score}_{\mathbf{w}}(x, y') + c(y, y')$
- 5 $\mathbf{w} \leftarrow \mathbf{w} + \eta(\phi(x, y) - \phi(x, \tilde{y}))$
- 6 Return \mathbf{w}

Optimization: Illustration

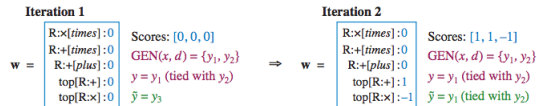
a Candidates GEN(x) for utterance $x = \text{two times two plus three}$



b Learning from logical forms (Section 4.1)



c Learning from denotations (Section 4.2)



Learning Model

- ▶ Components

- ▶ training data: $D = \{(x_i, y_i) | i \dots n\}$ ✓

Learning Model

- ▶ Components

- ▶ training data: $D = \{(x_i, y_i) | i \dots n\}$ ✓
- ▶ feature representation of data ✓

Learning Model

► Components

- training data: $D = \{(x_i, y_i) | i \dots n\}$ ✓
- feature representation of data ✓
- scoring and objective function ✓

Learning Model

► Components

- training data: $D = \{(x_i, y_i) | i \dots n\}$ ✓
- feature representation of data ✓
- scoring and objective function ✓
- optimization procedure ✓

Learning Model

► Components

- training data: $D = \{(x_i, y_i) | i \dots n\}$ ✓
- feature representation of data ✓
- scoring and objective function ✓
- optimization procedure ✓

► Important Ideas

- What kind of data do we learn from? (differs quite a bit)
- What kind of features do we need?

Experimentation and Evaluation

- ▶ **Training Set:** A portion of the data to train model on.
- ▶ **Test Set:** An unseen portion of the data to evaluate on.
- ▶ **Dev Set :** (optional) An unseen portion of the data for analysis, tuning hyper parameters, ..

Experimentation and Evaluation

- ▶ **Training Set:** A portion of the data to train model on.
- ▶ **Test Set:** An unseen portion of the data to evaluate on.
- ▶ **Dev Set :** (optional) An unseen portion of the data for analysis, tuning hyper parameters, ..

- ▶ **Evaluation1:** Given unseen examples, how often does my model produce the correct output semantic representation?
- ▶ **Evaluation2:** Given unseen examples, how often does my model produce the correct output answer?

Conclusions and Take Aways

- ▶ Presented a simple model that mixes machine learning and compositional semantics.
 - ▶ **Conceptually** describes most of the work in this class.
 - ▶ **Technically** describes many of the models we will use.
- ▶ **Fundamental Problem:** Which semantics representations do we use, and what do we learn from?

Conclusions and Take Aways

- ▶ Presented a simple model that mixes machine learning and compositional semantics.
 - ▶ **Conceptually** describes most of the work in this class.
 - ▶ **Technically** describes many of the models we will use.
- ▶ **Fundamental Problem:** Which semantics representations do we use, and what do we learn from?
- ▶ **Question:** Does this particular actually work?

Conclusions and Take Aways

- ▶ Presented a simple model that mixes machine learning and compositional semantics.
 - ▶ **Conceptually** describes most of the work in this class.
 - ▶ **Technically** describes many of the models we will use.
- ▶ **Fundamental Problem:** Which semantics representations do we use, and what do we learn from?
- ▶ **Question:** Does this particular actually work?
 - ▶ Yes! Liang et al. (2011) (lecture 5), Berant et al. (2013); Berant and Liang (2014) (presentation papers)

Roadmap

- ▶ **Lecture 2:** rule extraction, decoding (parsing perspective)
- ▶ **Lecture 3:** rule extraction, decoding (MT perspective)
- ▶ **Lecture 4:** structured classification and prediction.
- ▶ **Lecture 5:** grounded learning (**might skip**).

References I

- Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on Freebase from question-answer pairs. In *in Proceedings of EMNLP-2013*, pages 1533–1544.
- Berant, J. and Liang, P. (2014). Semantic parsing via paraphrasing. In *ACL (1)*, pages 1415–1425.
- Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.
- Liang, P., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of ACL-11*, pages 590–599.
- Mooney, R. (2008). Learning to connect language and perception. In *Proceedings of AAAI-2008*.
- Turney, P. D., Pantel, P., et al. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188.
- Woods, W. A. (1973). Progress in natural language understanding: an application to lunar geology. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, pages 441–450.