Lecture 2: Mixing Compositional Semantics and Machine Learning

Kyle Richardson

kyle@ims.uni-stuttgart.de

April 14, 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.*

\[
\text{john' } \rightarrow \text{“John”} \\
(\lambda x. (\text{study’ } x)) \rightarrow \text{“studies”}
\]
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.*

\[ \text{john'} \rightarrow "John" \]
\[ (\lambda x. (\text{study'} x)) \rightarrow "studies" \]

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

\[ \begin{array}{c}
\text{john'} \quad (\lambda x. (\text{study'} x)) \\
\downarrow \quad \downarrow \\
\text{John} \\
\text{studies}
\end{array} \]
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.*

\[\text{john}' \rightarrow \text{“John”} \]
\[\left(\lambda x. \text{(study’ } x)\right) \rightarrow \text{“studies”}\]

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

```python
>>> 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.*

\[
\begin{align*}
\text{john} & \rightarrow \text{“John”} \\
(\lambda x. (\text{study’ } x)) & \rightarrow \text{“studies”}
\end{align*}
\]

```python
>>> 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?
```

```text
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.*

\[
\text{john'} \rightarrow \text{"John"} \\
(\lambda x.(\text{study'} x)) \rightarrow \text{"studies"}
\]

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

```python
>>> 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.*

\[
\text{john' } \rightarrow \text{“John”} \\
(\lambda x. (\text{study’ } x)) \rightarrow \text{“studies”}
\]

```python
>>> 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
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.

\[
\text{john}' \rightarrow \text{"John"} \\
(\lambda x. (\text{study}' \ x)) \rightarrow \text{"studies"}
\]

```python
>>> 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.*

\[
\begin{align*}
\text{bill'} & \rightarrow "\text{Bill}" \\
(\lambda x.(\text{study'} x)) & \rightarrow "\text{study}" \\
(\lambda f. \lambda x.(\text{not } f x)) & \rightarrow "\text{does not}" 
\end{align*}
\]
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.

\[
\text{bill'} \rightarrow \text{“Bill”} \\
(\lambda x.(\text{study’ } x)) \rightarrow \text{“study”} \\
(\lambda f.\lambda x.(\text{not } (f \ x))) \rightarrow \text{“does not”}
\]

\[
(\lambda x.(\text{not } (\text{study’ } x)))(\text{bill})
\]

\[
\text{Bill} \ (\lambda f.\lambda x.(\text{not } (f \ x))) \ (\lambda x.(\text{study’ } x))
\]

\[
\text{does not} \quad \text{study}
\]
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.*

\[
\begin{align*}
bill' & \rightarrow "Bill" \\
(\lambda x. (\text{study'} x)) & \rightarrow "study" \\
(\lambda f. \lambda x. (\text{not} (f x))) & \rightarrow "does not"
\end{align*}
\]

```python
>>> 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.*

\[ \text{bill}' \rightarrow \text{"Bill"} \]
\[ (\lambda x. (\text{study}' \ x)) \rightarrow \text{"study"} \]
\[ (\lambda f. \lambda x. (\text{not} \ (f \ x))) \rightarrow \text{"does not"} \]

```python
g>>> students_studying = set(['john', 'mary'])
g>>> study = lambda x : x in students_studying
g>>> fun_application = lambda fun, val : fun(val)
g>>> neg = lambda F : (lambda x : not F(x))
g>>> 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.*

- bill’ $\rightarrow$ “Bill”
- $(\lambda x. (\text{study’ } x)) \rightarrow$ “study”
- $(\lambda f. \lambda x. (\text{not } (f x))) \rightarrow$ “does not”

```python
>>> 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.

```
bill' → "Bill"
(λx.(study' x)) → "study"
(λf.λx.(not (f x))) → "does not"
```

```python
>>> 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.*

\[
\begin{align*}
\text{bill'} & \longrightarrow \text{"Bill"} \\
(\lambda x.(\text{study'} \ x)) & \longrightarrow \text{"study"} \\
(\lambda f.\lambda x. (\text{not} \ (f \ x))) & \longrightarrow \text{"does not"}
\end{align*}
\]

```python
>>> 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.*

\[
\begin{align*}
\text{bill'} & \rightarrow \text{“Bill”} \\
(\lambda x.(\text{study’ } x)) & \rightarrow \text{“study”} \\
(\lambda f.\lambda x.(\text{not } (f \ x))) & \rightarrow \text{“does not”}
\end{align*}
\]

- 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.*

\[
\begin{align*}
\text{bill'} &\rightarrow \text{“Bill”} \\
(\lambda x. (\text{study’ } x)) &\rightarrow \text{“study”} \\
(\lambda f. \lambda x. (\text{not } (f x))) &\rightarrow \text{“does not”}
\end{align*}
\]

**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."

\[ \text{bill'} \rightarrow \text{“Bill”} \]
\[ (\lambda x. (\text{study’ } x)) \rightarrow \text{“study”} \]
\[ (\lambda f. \lambda x. (\text{not } (f \ x))) \rightarrow \text{“does not”} \]

**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.*

\[
\begin{align*}
\text{bill'} & \rightarrow \text{“Bill”} \\
(\lambda x. (\text{study’ } x)) & \rightarrow \text{“study”} \\
(\lambda f. \lambda x. (\text{not } (f x))) & \rightarrow \text{“does not”}
\end{align*}
\]

**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.*

\[
\begin{align*}
\text{bill'} & \rightarrow \text{“Bill”} \\
(\lambda x. (\text{study’ } x)) & \rightarrow \text{“study”} \\
(\lambda f. \lambda x. (\text{not } (f x))) & \rightarrow \text{“does not”}
\end{align*}
\]

**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)

<table>
<thead>
<tr>
<th>corpus</th>
<th>word-context matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <strong>furry</strong> dog is walking outside...</td>
<td>furry</td>
</tr>
<tr>
<td>The <strong>shiny</strong> car is driving...</td>
<td>dog</td>
</tr>
<tr>
<td>A <strong>furry</strong> cat is walking around...</td>
<td>cat</td>
</tr>
<tr>
<td>A <strong>shiny</strong> bike is driving...</td>
<td>car</td>
</tr>
<tr>
<td>....</td>
<td>bike</td>
</tr>
</tbody>
</table>
**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)

<table>
<thead>
<tr>
<th>corpus</th>
<th>word-context matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>furry</td>
<td>walking</td>
</tr>
<tr>
<td>dog</td>
<td>4</td>
</tr>
<tr>
<td>cat</td>
<td>3</td>
</tr>
<tr>
<td>car</td>
<td>0</td>
</tr>
<tr>
<td>bike</td>
<td>1</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>corpus</th>
<th>word-context matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>The furry dog is walking outside...</td>
<td>furry</td>
</tr>
<tr>
<td>The shiny cat is driving...</td>
<td>dog</td>
</tr>
<tr>
<td>A furry cat is walking around...</td>
<td>cat</td>
</tr>
<tr>
<td>A shiny bike is driving....</td>
<td>car</td>
</tr>
<tr>
<td>....</td>
<td>bike</td>
</tr>
</tbody>
</table>

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**

The **furry dog** is walking outside...

The **shiny car** is driving...

A **furry cat** is walking around...

A **shiny bike** is driving....

---

**word-context matrix**

<table>
<thead>
<tr>
<th></th>
<th>furry</th>
<th>walking</th>
<th>shiny</th>
<th>driving</th>
</tr>
</thead>
<tbody>
<tr>
<td>dog</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cat</td>
<td>12</td>
<td>25</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>car</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>bike</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>25</td>
</tr>
</tbody>
</table>

---

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)

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

**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-world 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:** \(<u, s, d>\)

- **u:** utterance
- **s:** semantic representation (symbolized as \(\hat{u}\))
- **d:** denotation (symbolized as \([s]\))
Compositional Model:

**Linguistic Objects:** \(< u, s, d >\)

- **u:** utterance
- **s:** semantic representation (symbolized as \(\hat{u}\))
- **d:** denotation (symbolized as \([s]\))

**Example:** \(< 'seven minus five', (- 7 5), 2 >\)
Compositional Model:

**Linguistic Objects:** $< u, s, d >$
- **u:** utterance
- **s:** semantic representation (symbolized as $\hat{u}$)
- **d:** denotation (symbolized as $[s]$)

**Example:**
- $< \text{seven minus five}', (- 7 5), 2 >$
- $< \text{two minus two times two}', (* (- 2 2) 2), 0 >$
Compositional Model:

Linguistic Objects: $< u, s, d >$

- $u$: utterance
- $s$: semantic representation (symbolized as $^u$)
- $d$: denotation (symbolized as $[s]$)

Example: $< 'seven minus five', (- 7 5), 2 >$
$< 'two minus two times two', (* (- 2 2) 2), 0 >$

semantic parsing: $u \rightarrow s$
Compositional Model:

**Linguistic Objects:** $< u, s, d >$

- **u:** utterance
- **s:** semantic representation (symbolized as $\hat{u}$)
- **d:** denotation (symbolized as $[s]$)

**Example:**

$<$ 'seven minus five', ($- 7 5$), $2 >$

$<$ 'two minus two times two', ($* (- 2 2) 2$), $0 >$

**Semantic Parsing:** $u \rightarrow s$

**Interpretation:** $s \rightarrow d$
Computational Modeling: The full picture

- Standard processing pipeline

List samples that contain every major element

\[
\text{List samples that contain every major element}
\]

\[
\text{Lunar QA system (Woods (1973))}
\]
Compositional Model: Context-free grammar

- provides the background grammar and interpretation rules

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantic representation</th>
<th>Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>N \rightarrow one</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>N \rightarrow two</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>R \rightarrow plus</td>
<td>+</td>
<td>the R such that $R(x, y) = x + y$</td>
</tr>
<tr>
<td>R \rightarrow minus</td>
<td>-</td>
<td>the R such that $R(x, y) = x - y$</td>
</tr>
<tr>
<td>R \rightarrow times</td>
<td>\times</td>
<td>the R such that $R(x, y) = x \times y$</td>
</tr>
<tr>
<td>S \rightarrow minus</td>
<td>-</td>
<td>the f such that $f(x) = -x$</td>
</tr>
<tr>
<td>N \rightarrow S N</td>
<td>[S][N]</td>
<td>$[S][N]$</td>
</tr>
<tr>
<td>N \rightarrow N_L R N_R</td>
<td>($R^{N_L}^{N_R}$)</td>
<td>$[R^{N_L}][R^{N_R}]$</td>
</tr>
</tbody>
</table>
Compositional Model: Context-free grammar

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

```
N: (plus (mult 2 2) 3)
```

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

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

```
plus(2, 2) # 4
```

```
N : 2 R : mult N : 2 plus N : 3
```

```
N : 2 R : times N : 2
```

```
N : 2 R : two
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
R : plus
```

```
N : 3
```

```
Compositional Model: Context-free grammar

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

![Tree Diagram]

\[
N: (\text{plus} (\text{mult} 2 2) 3)
\]

\[
\text{N : (mult 2 2)} \quad \text{R : plus} \quad \text{N : 3}
\]

\[
\text{N : 2} \quad \text{R : mul} \quad \text{N : 2} \quad \text{plus} \quad \text{three}
\]

\[
\text{two} \quad \text{times} \quad \text{two}
\]

\[
\text{>>> plus} = \text{lambda} \ x, y : x + y
\]

\[
\text{>>> mult} = \text{lambda} \ x, y : x * y
\]
Compositional Model: Context-free grammar

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

\[
N: (\text{plus} \ (\text{mult} \ 2 \ 2) \ 3)
\]

\[
N \ : \ (\text{mult} \ 2 \ 2) \quad R \ : \ \text{plus} \quad N \ : \ 3
\]

\[
N \ : \ 2 \quad R \ : \ \text{mult} \quad N \ : \ 2 \quad \text{plus} \quad \text{three}
\]

\[
\text{two} \quad \text{times} \quad \text{two}
\]

\[
>>> \ \text{plus} = \lambda x,y : x + y
\]

\[
>>> \ \text{mult} = \lambda x,y : x \times y
\]

\[
>>> \ \text{plus}(2,2) \neq 4
\]
Compositional Model: Context-free grammar

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

\[
N: (\text{plus} (\text{mult} 2 2) 3)
\]

\[
N: (\text{mult} 2 2) \quad \text{R: plus} \quad N: 3
\]

\[
N: 2 \quad \text{R: mult} \quad N: 2 \quad \text{plus} \quad \text{three}
\]

\[
\text{two} \quad \text{times} \quad \text{two}
\]

```python
>>> 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: \( u = \text{two times two plus three} \)

\[
\begin{align*}
N : (\text{plus} (\text{mult} 2 2) 3) \\
N : (\text{mult} 2 2) \quad R : \text{plus} \quad N : 3 \\
N : 2 \quad R : \text{mult} \quad N : 2 \quad \text{plus} \quad \text{three} \\
\text{two} \quad \text{times} \quad \text{two} \\
\end{align*}
\]

```python
>>> plus = lambda x,y : x + y
>>> mult = lambda x,y : x * y
>>> plus(mult(2,2),3) # 7
```

\[
\begin{align*}
N : (\text{plus} (\text{mult} 2 2) 3) \\
N : (\text{mult} 2 2) \quad R : \text{plus} \quad N : 3 \\
N : 2 \quad R : \text{mult} \quad N : 2 \quad \text{plus} \quad \text{three} \\
\text{two} \quad \text{times} \quad \text{two} \\
\end{align*}
\]
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)

- Example:  
  - \( u = \text{two times two plus three} \)
    - \( N: (\text{plus} (\text{mult} 2 2) 3) \)
    - \( R: \text{plus} \)
    - \( N: (\text{mult} 2 2) \)
    - \( R: \text{mult} \)
    - \( N: 2 \)
    - \( R: \text{times} \)
    - \( N: 2 \)

- Example:  
  - \( u = \text{two times two plus three} \)
    - \( N: (\text{plus} (\text{plus} 2 2) 3) \)
    - \( R: \text{plus} \)
    - \( N: (\text{plus} 2 2) \)
    - \( R: \text{plus} \)
    - \( N: 2 \)
    - \( R: \text{times} \)
    - \( N: 2 \)
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:** $u = \text{two times two plus three}$

![Diagram of tree structures for the example expression](image_url)
Compositional Model: Components

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

- **Issues:**
  - example: \( u = \text{two times two plus three} \)

\[
\begin{align*}
N: (\text{plus } (\text{mult } 2 2) 3) &\quad N: (\text{mult } 2 (\text{plus } 2 3)) \\
\text{N : (mult 2 2)} &\quad \text{R : plus N : 3} \\
\text{N : 2 R : mult N : 2} &\quad \text{N : 2 R : plus N : 3} \\
\text{two R : mult two} &\quad \text{two times three} \\
\end{align*}
\]
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 \( D = \{(x_i, y_i) | i \ldots n\} \)
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 $D = \{(x_i, y_i) | i \ldots n\}$
  - 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 $D = \{(x_i, y_i)|i...n\}$
  ▶ 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 \( D = \{(x_i, y_i) | i \ldots n\} \)
- feature representation of data
- scoring and objective function
- optimization procedure
Goal: Find the correct derivations and output using our compositional model
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)**
- \( (u = 'two minus two times two', s = (* (- 2 2) 2)) \)

**Denotations (less information)**
- \( (u = 'two minus two times two', r = 0) \)
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:** \( \text{(two times two plus three, (plus (mult 2 2) 3))} \)

![Diagram](attachment:image.png)

▶ **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)

<table>
<thead>
<tr>
<th></th>
<th>((x, y))</th>
<th>Feature representations (\phi(x, y))</th>
<th>'empty string'</th>
<th>'last word'</th>
<th>'all words'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td></td>
<td></td>
<td>(\epsilon)</td>
<td>five</td>
<td>([\text{twenty, five}])</td>
</tr>
<tr>
<td></td>
<td>(twenty-five, O)</td>
<td></td>
<td>(\epsilon)</td>
<td>one</td>
<td>([\text{thirty, one}])</td>
</tr>
<tr>
<td></td>
<td>(thirty-one, O)</td>
<td></td>
<td>(\epsilon)</td>
<td>nine</td>
<td>([\text{forty, nine}])</td>
</tr>
<tr>
<td></td>
<td>(forty-nine, O)</td>
<td></td>
<td>(\epsilon)</td>
<td>two</td>
<td>([\text{fifty, two}])</td>
</tr>
<tr>
<td></td>
<td>(fifty-two, E)</td>
<td></td>
<td>(\epsilon)</td>
<td>two</td>
<td>([\text{eighty, two}])</td>
</tr>
<tr>
<td></td>
<td>(eighty-two, E)</td>
<td></td>
<td>(\epsilon)</td>
<td>four</td>
<td>([\text{eighty, four}])</td>
</tr>
<tr>
<td></td>
<td>(eighty-four, E)</td>
<td></td>
<td>(\epsilon)</td>
<td>six</td>
<td>([\text{eighty, six}])</td>
</tr>
<tr>
<td>Test</td>
<td>(eighty-five, O)</td>
<td></td>
<td>(\epsilon \rightarrow \text{E})</td>
<td>five \rightarrow \text{O}</td>
<td>([\text{eighty, five} \rightarrow \text{E}])</td>
</tr>
</tbody>
</table>
"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

<table>
<thead>
<tr>
<th></th>
<th>$(x, y)$</th>
<th>Feature representations $\phi(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>‘empty string’</td>
</tr>
<tr>
<td>Train</td>
<td>(twenty-five, O)</td>
<td>$\epsilon$ five</td>
</tr>
<tr>
<td></td>
<td>(thirty-one, O)</td>
<td>$\epsilon$ one</td>
</tr>
<tr>
<td></td>
<td>(forty-nine, O)</td>
<td>$\epsilon$ nine</td>
</tr>
<tr>
<td></td>
<td>(fifty-two, E)</td>
<td>$\epsilon$ two</td>
</tr>
<tr>
<td></td>
<td>(eighty-two, E)</td>
<td>$\epsilon$ two</td>
</tr>
<tr>
<td></td>
<td>(eighty-four, E)</td>
<td>$\epsilon$ four</td>
</tr>
<tr>
<td></td>
<td>(eighty-six, E)</td>
<td>$\epsilon$ six</td>
</tr>
<tr>
<td>Test</td>
<td>(eighty-five, O)</td>
<td>$\epsilon \rightarrow E$ five $\rightarrow O$</td>
</tr>
</tbody>
</table>
**Feature Extraction Example**

**input:** \( x = \text{two times two plus three} \).

\[
y_1 = N: (\text{plus } (\text{mult } 2 2)\ 3)
\]

\[
y_2 = N: (\text{plus } (\text{plus } 2 2)\ 3)
\]

\[
\phi(x,y_1) = \begin{align*}
R : \text{mult } \rightarrow 1 \\
R : \text{plus } \rightarrow 1 \\
top [ R : \text{plus } ] \rightarrow 1 \\
\ldots
\end{align*}
\]

\[
\phi(x,y_2) = \begin{align*}
R : \text{plus } \rightarrow 1 \\
R : \text{mult } \rightarrow 1 \\
top [ R : \text{plus } ] \rightarrow 1 \\
\ldots
\end{align*}
\]
Scoring Function

(Linear) Score Function

\[
\text{Score}_w(x,y) = w \cdot \phi(x, y) = \sum_{j=1}^{d} w_j \phi(x, y)
\]
(Linear) Score Function

- Score\textsubscript{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 \ ...]
**Scoring Function**

(Linear) Score Function

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

\[
\phi(x, y_2) = \begin{array}{c}
w_1 \text{ R : plus [ 'times' ] } \rightarrow 1 \\
w_2 \text{ R : plus [ 'plus' ] } \rightarrow 1 \\
w_3 \text{ top [ R : plus ] } \rightarrow 1 \\
... 
\end{array}
\]

\[
score_w(x, y_2) = w \cdot \phi(x, y_2) = (0.1 \times 1.0) + (0.2 \times 1.0) + (0.0 \times 1.0)
\]
Scoring Function

(Linear) Score Function

- 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 ...]
- prediction: arg\-\max_{y \in Y} 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.

\[ y_1 = N: (\text{plus} (\text{mult} 2 2) 3) \]

\[ \phi(x,y_1) = \begin{cases} 
R : \text{mult} & [ 'times' ] \rightarrow 1 \\
R : \text{plus} & [ 'plus' ] \rightarrow 1 \\
top & [ R : \text{plus} ] \rightarrow 1 \\
\end{cases} \]

\[ y_2 = N: (\text{plus} (\text{plus} 2 2) 3) \]

\[ \phi(x,y_2) = \begin{cases} 
R : \text{plus} & [ 'times' ] \rightarrow 1 \\
R : \text{plus} & [ 'plus' ] \rightarrow 1 \\
top & [ R : \text{plus} ] \rightarrow 1 \\
\end{cases} \]
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.

\[ y_1 = N: (\text{plus} \ (\text{mult} \ 2 \ 2) \ 3) \]

\[
\begin{array}{c}
N: (\text{mult} \ 2 \ 2) \\
\quad R: \text{plus} \quad N: 3 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad N: 2 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad R: \text{mult} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{two} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{times} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{two} \\
\end{array}
\]

\[
\begin{array}{c}
\phi(x, y_1) = \\
\quad R: \text{mult} \ [ \text{times} ] \to 1 \\
\quad R: \text{plus} \ [ \text{plus} ] \to 1 \\
\quad \text{plus} \ [ R: \text{mult} ] \to 1 \\
\quad \ldots \\
\end{array}
\]

\[ N: (\text{mult} \ 2 \ (\text{plus} \ 2 \ 3)) \]

\[
\begin{array}{c}
N: 2 \\
\quad R: \text{mult} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad N: 2 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad R: \text{plus} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{two} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{times} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{two} \\
\end{array}
\]

\[
\begin{array}{c}
\phi(x, y_2) = \\
\quad R: \text{plus} \ [ \text{times} ] \to 1 \\
\quad R: \text{plus} \ [ \text{plus} ] \to 1 \\
\quad \text{mult} \ [ R: \text{plus} ] \to 1 \\
\quad \ldots \\
\end{array}
\]
Objectives: What do we want to learn? (formal)

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

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

- ('two minus two times two', \(s = (* (- 2 2) 2)\))
Objectives: What do we want to learn? (formal)

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

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

- ('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)

\[
\min_{w \in \mathbb{R}^d} \sum_{(x, y) \in D} \max_{y' \in Y} \left[ \text{Score}_w(x, y') + c(y, y') \right] - \text{Score}_w(x, y)
\]

▶ ('two minus two times two', \(s = (\ast (- 2 2) 2)\))

▶ **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(D, T, η)

D: a set of training examples (x, y) ∈ (X × Y)
T: the number of passes to make through the data
η > 0: learning rate (e.g., \( \frac{1}{\sqrt{T}} \))

1. Initialize w ← 0
2. Repeat T times
3. for each (x, y) ∈ D (in random order)
4.     ŷ ← arg max_{y' ∈ Y} Score_w(x, y') + c(y, y')
5.     w ← w + η(φ(x, y) − φ(x, ŷ))
6. Return w
```
Optimization: Illustration

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

\[
\begin{align*}
\phi(x, y_1) &= \begin{cases} R:x[\text{times}]:1, \\ R:+[\text{plus}]:1, \\ \text{top}[R:+]:1 \end{cases} \\
\phi(x, y_2) &= \begin{cases} R:+[\text{plus}]:1, \\ \text{top}[R:+]:1 \end{cases} \\
\phi(x, y_3) &= \begin{cases} R:x[\text{times}]:1, \\ R:+[\text{plus}]:1, \\ \text{top}[R:+]:1 \end{cases}
\end{align*}
\]

**b** Learning from logical forms (Section 4.1)

**Iteration 1**

\[
\begin{align*}
w &= \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:0, \\ \text{top}[R:x]:0 \end{cases} \\
y &= y_1 \\
\tilde{y} &= y_3 \text{ (tied with } y_2 \text{)}
\end{align*}
\]

**Scores:** \([0, 0, 0]\)

\[
\Rightarrow \quad w = \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:0, \\ \text{top}[R:x]:0 \end{cases} \\
y &= y_1
\]

**Iteration 2**

\[
\begin{align*}
w &= \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:1, \\ \text{top}[R:x]:1 \end{cases} \\
y &= y_1
\end{align*}
\]

**Scores:** \([1, 1, -1]\)

\[
\Rightarrow \quad w = \begin{cases} R:x[\text{times}]:1, \\ R:+[\text{times}]:-1, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:1, \\ \text{top}[R:x]:-1 \end{cases} \\
y &= y_1, \tilde{y} = y_2
\]

**Iteration 3**

\[
\begin{align*}
w &= \begin{cases} R:x[\text{times}]:1, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:0, \\ \text{top}[R:x]:-1 \end{cases} \\
y &= y_1
\end{align*}
\]

**Scores:** \([2, 0, 0]\)

**C** Learning from denotations (Section 4.2)

**Iteration 1**

\[
\begin{align*}
w &= \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:0, \\ \text{top}[R:x]:0 \end{cases} \\
\text{GEN}(x, d) &= \{y_1, y_2\} \\
y &= y_1 \text{ (tied with } y_2 \text{)} \\
\tilde{y} &= y_3
\end{align*}
\]

\[
\Rightarrow \quad w = \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:0, \\ \text{top}[R:x]:0 \end{cases} \\
\text{GEN}(x, d) &= \{y_1, y_2\} \\
y &= y_1 \text{ (tied with } y_2 \text{)} \\
\tilde{y} &= y_3
\]

**Scores:** \([0, 0, 0]\)

**Iteration 2**

\[
\begin{align*}
w &= \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:1, \\ \text{top}[R:x]:-1 \end{cases} \\
\text{GEN}(x, d) &= \{y_1, y_2\} \\
y &= y_1 \text{ (tied with } y_2 \text{)} \\
\tilde{y} &= y_3
\end{align*}
\]

**Scores:** \([1, 1, -1]\)

\[
\Rightarrow \quad w = \begin{cases} R:x[\text{times}]:0, \\ R:+[\text{times}]:0, \\ R:+[\text{plus}]:0, \\ \text{top}[R:+]:1, \\ \text{top}[R:x]:-1 \end{cases} \\
\text{GEN}(x, d) &= \{y_1, y_2\} \\
y &= y_1 \text{ (tied with } y_2 \text{)} \\
\tilde{y} &= y_1 \text{ (tied with } y_2 \text{)}
\]

**Scores:** \([1, 1, -1]\)
Learning Model

- Components
  - training data: $D = \{(x_i, y_i) | i \ldots n\}$ ✓
Learning Model

- Components
  - training data: $D = \{(x_i, y_i) | i \ldots n\}$ ✓
  - feature representation of data ✓
Learning Model

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

- Components
  - training data: $D = \{(x_i, y_i) | i=1...n\}$ ✓
  - 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?
Learning Model

- **Components**
  - training data: $D = \{(x_i, y_i) | i \ldots n\}$ ✔
  - 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).


