Project Proposal

• project proposal due today
Midterm

• midterm in one week
• you can bring notes
  – but we’ll try to give you all formulas/definitions you’ll need
• Monday: review for midterm, including sample questions
Roadmap

• words, morphology, lexical semantics
• text classification
• language modeling
• word embeddings
• recurrent/recursive/convolutional networks in NLP
• sequence labeling, HMMs, dynamic programming
• syntax and syntactic parsing
• semantics, compositionality, semantic parsing
• machine translation and other NLP tasks
What is Syntax?

• rules, principles, processes that govern sentence structure of a language
Constituent Parse (Bracketing/Tree)

(S (NP the man) (VP walked (PP to (NP the park))))

the man walked to the park

Key:
S = sentence
NP = noun phrase
VP = verb phrase
PP = prepositional phrase
DT = determiner
NN = noun
VBD = verb (past tense)
IN = preposition
(S (NP the man) (VP walked (PP to (NP the park))))

 Constituent Parse


nonterminals

preterminals

terminals

deep structure: (S (NP the man) (VP walked (PP to (NP the park)))))
Attachment Ambiguity

One morning I shot an elephant in my pajamas. How he got into my pajamas I'll never know.

Groucho Marx
American Comedian
1890 - 1977
NLP Task: Constituent Parsing

• given a sentence, output its constituent parse
• widely-studied task with a rich history
• most based on the Penn Treebank (Marcus et al.), developed at Penn in early 1990s

• Treebank = “corpus of annotated parse trees”
Context-Free Grammar (CFG)

• has “rewrite rules” to rewrite nonterminals as terminals or other nonterminals

S → NP  VP

“S goes to NP  VP”

NP → DT  NN

VP → VBD  PP

PP → IN  NP

NN → man

DT → the
Context-Free Grammar (CFG)

- sequence of rewrites corresponds to a bracketing (induces a hierarchical tree structure)

```
the man walked to the park
```

```
S → NP VP
NP → DT NN
VP → VBD PP
```

```
VBD → walked
IN → to
```

```
the → DT
```

```
the Park → NN
```

```
the man walked to the park
```
Why “context-free”? 

• a rule to rewrite NP does not depend on the context of NP 

• that is, the left-hand side of a rule is only a single non-terminal (without any other context)
Probabilistic Context-Free Grammar (PCFG)

• assign probabilities to rewrite rules:

  NP → DT NN  0.5
  NP → NNS    0.3
  NP → NP PP  0.2

same nonterminal can be on both left and right sides
Probabilistic Context-Free Grammar (PCFG)

• assign probabilities to rewrite rules:

\[
\begin{align*}
\text{NP} & \rightarrow \text{DT} \ \text{NN} \quad 0.5 \\
\text{NP} & \rightarrow \text{NNS} \quad 0.3 \\
\text{NP} & \rightarrow \text{NP} \ \text{PP} \quad 0.2
\end{align*}
\]

probabilities must sum to one for each left-hand side nonterminal
Probabilistic Context-Free Grammar (PCFG)

• assign probabilities to rewrite rules:

\[
\begin{align*}
\text{NP} & \rightarrow \text{DT} \quad \text{NN} \quad 0.5 \\
\text{NP} & \rightarrow \text{NNS} \quad 0.3 \\
\text{NP} & \rightarrow \text{NP} \quad \text{PP} \quad 0.2 \\
\text{NN} & \rightarrow \text{man} \quad 0.01 \\
\text{NN} & \rightarrow \text{park} \quad 0.0004 \\
\text{NN} & \rightarrow \text{walk} \quad 0.002 \\
\text{NN} & \rightarrow \text{...} \\
\end{align*}
\]

given a treebank, we can estimate these probabilities using maximum likelihood estimation ("count and normalize")

just like n-gram language models and HMMs for POS tagging
Probabilistic Context-Free Grammar (PCFG)

• for each nonterminal, a PCFG has a probability distribution over possible right-hand side sequences

• so, a PCFG assigns probabilities to:
  – bracketings of sentences
  – sequences of rewrite operations (derivations) that eventually terminate in terminals
  – hierarchical tree structures that ground out in sequences of terminals

• these are different ways of saying the same thing
Constituent Parsing

- **evaluation**: `evalb score`
  - first compute precision and recall (at the level of constituents)
  - then compute F1 (harmonic mean of precision and recall)
Precision, Recall, F1

• **precision:**
  – what fraction of the things I found are good?

\[ P = \frac{|F \cap G|}{|F|} \]

• **recall:**
  – what fraction of good things did I find?

\[ R = \frac{|F \cap G|}{|G|} \]

• **F1 score:**
  – harmonic mean of precision and recall
Modeling, Inference, Learning

**inference:** solve $\text{argmax}$

classify$(x, w) = \text{argmax} \ y$

**modeling:** define score function

score$(x, y, w)$

**learning:** choose $w$

- $x = \text{a sentence}$
- $y = \text{a constituent parse}$
- inference requires iterating over all possible constituent parses!
- this can be done using dynamic programming but is still expensive (cubic in the sentence length)
Inference in PCFGs

• to find max-probability tree for a sentence, use dynamic programming: **CKY algorithm**

• to find the best way to build a tree covering words $i$ to $j$:
  
  – consider all possible “split points” $k$ between $i$ and $j$
  
  – for each split point $k$, consider all possible nonterminals for the two smaller trees created by that split
There near the top of the list is quarterback Troy Aikman.
There near the top of the list is quarterback Troy Aikman.

**CKY Algorithm**

\[
C(Z, i, j) = \max_k \max_{A,B} (C(A, i, k) C(B, k, j) \text{score}(\langle Z \rightarrow AB \rangle))
\]

max probability of all ways to build a constituent with nonterminal \( Z \) from \( i \) to \( j \)
There near the top of the list is quarterback Troy Aikman.
• detail: CKY requires the PCFG to be in **Chomsky Normal Form (CNF)**
• basically: every rule has either 2 nonterminals or 1 terminal on the right-hand side
How well does a PCFG work?

- a PCFG learned from the Penn Treebank with maximum likelihood estimation (count & normalize) gets about 73% F1 score
- state-of-the-art parsers are around 92%
How well does a PCFG work?

• a PCFG learned from the Penn Treebank with maximum likelihood estimation (count & normalize) gets about 73% F1 score
• state-of-the-art parsers are around 92%
• but, simple modifications can improve the PCFG a lot!
  – smoothing
  – tree transformations (selective flattening)
  – “parent annotation”
Parent Annotation

$VP \rightarrow V \ NP \ PP$

$VP^S \rightarrow V \ NP^{VP} \ PP^{VP}$

adds more information, but also fragments counts, making parameter estimates noisier (since we’re just using MLE)
PCFG Models of Linguistic Tree Representations

Mark Johnson*
Brown University

The kinds of tree representations used in a treebank corpus can have a dramatic effect on performance of a parser based on the PCFG estimated from that corpus, causing the estimated likelihood of a tree to differ substantially from its frequency in the training corpus. This paper points out that the Penn II treebank representations are of the kind predicted to have such an effect, and describes a simple node relabeling transformation that improves a treebank PCFG-based parser’s average precision and recall by around 8%, or approximately half of the performance difference between a simple PCFG model and the best broad-coverage parsers available today. This performance variation comes about because any PCFG, and hence the corpus of trees from which the PCFG is induced, embodies independence assumptions about the distribution of words and phrases. The particular independence assumptions implicit in a tree representation can be studied theoretically and investigated empirically by means of a tree transformation/detransformation process.
Johnson (1998)

<table>
<thead>
<tr>
<th></th>
<th>22</th>
<th>22 Id</th>
<th>Id</th>
<th>NP-VP</th>
<th>N'-V'</th>
<th>Flatten</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of rules</td>
<td></td>
<td>2,269</td>
<td>14,962</td>
<td>14,297</td>
<td>14,697</td>
<td>22,652</td>
<td>22,773</td>
</tr>
<tr>
<td>Precision</td>
<td>1</td>
<td>0.772</td>
<td>0.735</td>
<td>0.730</td>
<td>0.735</td>
<td>0.745</td>
<td>0.800</td>
</tr>
<tr>
<td>Recall</td>
<td>1</td>
<td>0.728</td>
<td>0.697</td>
<td>0.705</td>
<td>0.701</td>
<td>0.723</td>
<td>0.792</td>
</tr>
<tr>
<td>NP attachments</td>
<td>279</td>
<td>0</td>
<td>67</td>
<td>330</td>
<td>69</td>
<td>154</td>
<td>217</td>
</tr>
<tr>
<td>VP attachments</td>
<td>299</td>
<td>424</td>
<td>384</td>
<td>0</td>
<td>503</td>
<td>392</td>
<td>351</td>
</tr>
<tr>
<td>NP* attachments</td>
<td>339</td>
<td>3</td>
<td>67</td>
<td>399</td>
<td>69</td>
<td>161</td>
<td>223</td>
</tr>
<tr>
<td>VP* attachments</td>
<td>412</td>
<td>668</td>
<td>662</td>
<td>150</td>
<td>643</td>
<td>509</td>
<td>462</td>
</tr>
</tbody>
</table>
How well does a PCFG work?

- PCFG learned from the Penn Treebank with MLE gets about 73% F1 score
- state-of-the-art parsers are around 92%
- simple modifications can improve PCFGs:
  - smoothing
  - tree transformations (selective flattening)
  - parent annotation
How well does a PCFG work?

- PCFG learned from the Penn Treebank with MLE gets about 73% F1 score
- state-of-the-art parsers are around 92%
- simple modifications can improve PCFGs:
  - smoothing
  - tree transformations (selective flattening)
  - parent annotation
  - lexicalization
Three Generative, Lexicalised Models for Statistical Parsing

Michael Collins*
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, 19104, U.S.A.
mcollins@gradient.cis.upenn.edu

Abstract
In this paper we first propose a new statistical parsing model, which is a generative model of lexicalised context-free grammar. We then extend the model to include a probabilistic treatment of both subcategorisation and wh-movement. Results on Wall Street Journal text show that the parser performs at 88.1/87.5% constituent precision/recall, an average improvement of 2.3% over (Collins 96).

1 Introduction
Generative models of syntax have been central in linguistics since they were introduced in (Chom-
Lexicalized PCFGs

nonterminals are decorated with the head word of the subtree
Lexicalization

• this adds a lot more rules!
• many more parameters to estimate → smoothing becomes much more important
  – e.g., right-hand side of rule might be factored into several steps
• but it’s worth it because head words are really useful for constituent parsing
## Results (Collins, 1997)

<table>
<thead>
<tr>
<th>MODEL</th>
<th>≤ 40 Words (2245 sentences)</th>
<th>LR</th>
<th>LP</th>
<th>CBs</th>
<th>0 CBs</th>
<th>≤ 2 CBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Magerman 95)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Collins 96)</td>
<td></td>
<td>84.6%</td>
<td>84.9%</td>
<td>1.26</td>
<td>56.6%</td>
<td>81.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85.8%</td>
<td>86.3%</td>
<td>1.14</td>
<td>59.9%</td>
<td>83.6%</td>
</tr>
<tr>
<td>Model 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model 2</td>
<td></td>
<td>87.4%</td>
<td>88.1%</td>
<td>0.96</td>
<td>65.7%</td>
<td>86.3%</td>
</tr>
<tr>
<td>Model 3</td>
<td></td>
<td>88.1%</td>
<td>88.6%</td>
<td>0.91</td>
<td>66.5%</td>
<td>86.9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>88.1%</td>
<td>88.6%</td>
<td>0.91</td>
<td>66.4%</td>
<td>86.9%</td>
</tr>
</tbody>
</table>
Head Rules

• how are heads decided?
• initially, researchers used deterministic head rules (Magerman/Collins)
• for a PCFG rule $A \rightarrow B_1 \ldots B_N$, these head rules say which of $B_1 \ldots B_N$ is the head of the rule
• examples:
  
  $S \rightarrow NP \ VP$

  $VP \rightarrow VBD \ NP \ PP$

  $NP \rightarrow DT \ JJ \ NN$
Head Annotation

Heads have bold outline

e.g., VP is head of S -> NP VP

from Noah Smith
Lexical Head Annotation

propagate lexical heads up the tree

from Noah Smith
Lexical Head Annotation $\rightarrow$ Dependencies

remove nonlexical parts:

from Noah Smith
Dependencies

merge redundant nodes:

from Noah Smith
constituent parse:

dependency parse:
constituent parse:

labeled dependency parse:

cnst = “nominal subject”
dobj = “direct object”
prep = “preposition modifier”
pobj = “object of preposition”
det = “determiner”
constituent parse:

captures some semantic relationships

nsubj = “nominal subject”
dobj = “direct object”
prep = “preposition modifier”
pobj = “object of preposition”
det = “determiner”
A Typed Dependency Tree

I prefer the morning flight through Denver
Some Dependency Relations

<table>
<thead>
<tr>
<th>Clausal Argument Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSUBJ</td>
<td>Nominal subject</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Direct object</td>
</tr>
<tr>
<td>IOBJ</td>
<td>Indirect object</td>
</tr>
<tr>
<td>CCOMP</td>
<td>Clausal complement</td>
</tr>
<tr>
<td>XCOMP</td>
<td>Open clausal complement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nominal Modifier Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMOD</td>
<td>Nominal modifier</td>
</tr>
<tr>
<td>AMOD</td>
<td>Adjectival modifier</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Numeric modifier</td>
</tr>
<tr>
<td>APPOS</td>
<td>Appositional modifier</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner</td>
</tr>
<tr>
<td>CASE</td>
<td>Prepositions, postpositions and other case markers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Notable Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONJ</td>
<td>Conjunct</td>
</tr>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
</tbody>
</table>

*Figure 14.2* Selected dependency relations from the Universal Dependency set. *(de Marneffe et al., 2014)*
## Some Dependency Relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Examples with <em>head</em> and <em>dependent</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>NSUBJ</td>
<td>United canceled the flight.</td>
</tr>
<tr>
<td>DOBJ</td>
<td>United diverted the <strong>flight</strong> to Reno.</td>
</tr>
<tr>
<td></td>
<td>We <em>booked</em> her the first <strong>flight</strong> to Miami.</td>
</tr>
<tr>
<td>IOBJ</td>
<td>We <em>booked</em> her the flight to Miami.</td>
</tr>
<tr>
<td>NMOD</td>
<td>We took the <strong>morning flight</strong>.</td>
</tr>
<tr>
<td>AMOD</td>
<td>Book the <strong>cheapest flight</strong>.</td>
</tr>
<tr>
<td>NUMMOD</td>
<td>Before the storm JetBlue canceled 1000 <strong>flights</strong>.</td>
</tr>
<tr>
<td>APPOS</td>
<td>United, a <strong>unit</strong> of UAL, matched the fares.</td>
</tr>
<tr>
<td>DET</td>
<td>The <strong>flight</strong> was canceled.</td>
</tr>
<tr>
<td></td>
<td>Which <strong>flight</strong> was delayed?</td>
</tr>
<tr>
<td>CONJ</td>
<td>We flew to Denver and <strong>drove</strong> to Steamboat.</td>
</tr>
<tr>
<td>CC</td>
<td>We flew to Denver <strong>and drove</strong> to Steamboat.</td>
</tr>
<tr>
<td>CASE</td>
<td>Book the flight <strong>through</strong> Houston.</td>
</tr>
</tbody>
</table>

---

*Figure 14.3* Examples of core Universal Dependency relations.
Crossing Dependencies = Nonprojective Tree

if dependencies cross ("nonprojective"), no longer corresponds to a CFG

JetBlue canceled our flight this morning which was already late
Projective vs. Nonprojective Dependencies

• English dependency treebanks are mostly projective
  – but when focusing more on semantic relationships, often becomes more nonprojective
• some (relatively) free word order languages, like Czech, are fairly nonprojective
Annotating Dependencies

• for many years, researchers build dependency parsers from deterministic head rules
• deterministic head rules are a blunt instrument
• would be better to directly annotate dependencies!
• there have been many annotation efforts with this goal
Universal Dependencies (UD) is a project that is developing cross-linguistically consistent treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective.

This is illustrated in the following parallel examples from English, Bulgarian, Czech and Swedish, where the main grammatical relations involving a passive verb, a nominal subject and an oblique agent are the same, but where the concrete grammatical realization varies.

1. The dog was chased by the cat.
2. Кучето се преследваше от котката.
4. Hunden jagades av katten.
Dependency Parsing

• several widely-used algorithms
• different guarantees but similar performance in practice
• graph-based:
  – dynamic programming (Eisner, 1997)
  – minimum spanning tree (McDonald et al., 2005)
• transition-based:
  – shift-reduce (Nivre, *inter alia*)
Transition-Based Dependency Parsing

**Figure 14.5** Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.
Transition-Based Parsing

• there are many variations of greedy parsers that build parse structures as they process a sentence from left to right
  – “shift-reduce”, “transition-based”, etc.
• these form the backbone of many modern neural dependency (and constituency!) parsers
• we’ll go through an example (thanks to Noah Smith for these slides)
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Greedy Parsing with a Stack

Stack:

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Greedy Parsing with a Stack

Stack:

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Greedy Parsing with a Stack

Stack:

Buffer:

Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
Many millennials, of which 50 percent are estimated to have voted, say political parties must listen to their concerns to get support.
<table>
<thead>
<tr>
<th>Stack&lt;sub&gt;t&lt;/sub&gt;</th>
<th>Buffer&lt;sub&gt;t&lt;/sub&gt;</th>
<th>Action</th>
<th>Stack&lt;sub&gt;t+1&lt;/sub&gt;</th>
<th>Buffer&lt;sub&gt;t+1&lt;/sub&gt;</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u, u), (v, v), S</td>
<td>B</td>
<td>REDUCE-RIGHT&lt;sub&gt;r&lt;/sub&gt;</td>
<td>(g&lt;sub&gt;r&lt;/sub&gt;(u, v), u), S</td>
<td>B</td>
<td>u → v</td>
</tr>
<tr>
<td>(u, u), (v, v), S</td>
<td>B</td>
<td>REDUCE-LEFT&lt;sub&gt;r&lt;/sub&gt;k</td>
<td>(g&lt;sub&gt;r&lt;/sub&gt;(v, u), v), S</td>
<td>B</td>
<td>u ← v</td>
</tr>
<tr>
<td>S</td>
<td>(u, u), B</td>
<td>SHIFT</td>
<td>(u, u), S</td>
<td>B</td>
<td>—</td>
</tr>
</tbody>
</table>

Figure 3: Parser transitions indicating the action applied to the stack and buffer and the resulting stack and buffer states. Bold symbols indicate (learned) embeddings of words and relations, script symbols indicate the corresponding words and relations.
• Early work used multi-class linear classifiers to output a parsing decision (shift, reduce-left, or reduce-right)
• Chen et al. (2014) used a feed-forward network for this
• Dyer et al. (2015) used RNNs to model the history of parsing decisions, the partial parses so far (the “stack”), and the sentence
Stack RNNs

Figure 1: A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as TOP in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (right). The boxes in the lowest rows represent stack contents, which are the inputs to the LSTM, the upper rows are the outputs of the LSTM (in this paper, only the output pointed to by TOP is ever accessed), and the middle rows are the memory cells (the $c_t$'s and $h_t$'s) and gates. Arrows represent function applications (usually affine transformations followed by a nonlinearity), refer to §2.1 for specifics.

Dyer et al. (ACL 2015)
Figure 2: Parser state computation encountered while parsing the sentence “an overhasty decision was made.” Here $S$ designates the stack of partially constructed dependency subtrees and its LSTM encoding; $B$ is the buffer of words remaining to be processed and its LSTM encoding; and $A$ is the stack representing the history of actions taken by the parser. These are linearly transformed, passed through a ReLU nonlinearity to produce the parser state embedding $p_t$. An affine transformation of this embedding is passed to a softmax layer to give a distribution over parsing decisions that can be taken.

Dyer et al. (ACL 2015)
Many millennials of which are estimated to have voted...
Dependency Parsers

- Stanford parser
- TurboParser
- Joakim Nivre’s MALT parser
- Ryan McDonald’s MST parser
- and many others for many non-English languages
Projective vs. Nonprojective Dependency Parsing

• nonprojective parsing can be formulated as a minimum spanning tree problem

• projective parsing cannot, but dynamic programming algorithms can be used (variations of CKY) as well as transition-based parsers
Complexity Comparison

• constituent parsing: $O(Gn^3)$
  – parsing complexity depends on grammar structure
    (“grammar constant” $G$)
  – since it has lots of nonterminal-only rules at the top of
    the tree, there are many rule probabilities to estimate

• dependency parsing: $O(n^3)$
  – operates directly on words, so parsing complexity has
    no grammar constant
  – features designed on possible dependencies (pairs of
    words) and larger structures
  – transition-based parsing algorithms are $O(n)$, though
    not optimal; also, non-projective parsing is faster
Applications of Dependency Parsing

• widely used for NLP tasks because:
  – faster than constituent parsing
  – captures more semantic information
• text classification (features on dependencies)
• syntax-based machine translation
• relation extraction
  – e.g., extract relation between Sam Smith and AITech: *Sam Smith was named new CEO of AITech.*
  – use dependency path between *Sam Smith* and *AITech*:
    • Smith $\rightarrow$ named, named $\leftarrow$ CEO, CEO $\leftarrow$ of, of $\leftarrow$ AITech
Summary: two types of grammars

• phrase structure / constituent grammars
  – inspired mostly by Chomsky and others
  – only appropriate for certain languages (e.g., English)

• dependency grammars
  – closer to a semantic representation; some have made this more explicit
  – problematic for certain syntactic structures (e.g., conjunctions, nesting of noun phrases, etc.)

• both are widely used in NLP
• you can find constituent parsers and dependency parsers for several languages online
Recursive Neural Networks for NLP

\[ x = \textit{it fell apart} \]

- run a syntactic parser on the sentence
- construct vector recursively at each split point:
Recursive Neural Networks for NLP

\[ x = \textit{it fell apart} \]

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

\[ h_1 = \text{emb}(\text{it}) \]

\[ \text{emb}(\text{it}) \]

\[ h_2 \]

\[ \text{emb}(\text{fell}) \]

\[ h_3 \]

\[ \text{emb}(\text{apart}) \]
Recursive Neural Networks for NLP

\[ \mathbf{x} = \text{it fell apart} \]

- run a syntactic parser on the sentence
- construct vector recursively at each split point:
Recursive Neural Networks for NLP

\[ \mathbf{x} = \textit{it fell apart} \]

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

\[
\begin{align*}
\mathbf{h}_5 &= g \left( \mathbf{W} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_4 \end{bmatrix} + \mathbf{b} \right) \\
\mathbf{h}_4 &= g \left( \mathbf{W} \begin{bmatrix} \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix} + \mathbf{b} \right)
\end{align*}
\]
Recursive Neural Networks for NLP

- same parameters used at every split point
- order of children matters (different weights used for left and right child)

\[ h_5 = g \left( W \begin{bmatrix} h_1 \\ h_4 \end{bmatrix} + b \right) \]
\[ h_4 = g \left( W \begin{bmatrix} h_2 \\ h_3 \end{bmatrix} + b \right) \]