Combinatory
Categorial Grammar
Combinatory Categorial Grammar (CCG)

- Categorial grammar (CG) is one of the oldest grammar formalisms
- *Combinatory* Categorial Grammar now well established and computationally well founded (Steedman, 1996, 2000)
- Account of syntax; semantics; prodody and information structure; automatic parsers; generation
Combinatory Categorial Grammar (CCG)

- CCG is a lexicalized grammar
- An elementary syntactic structure – for CCG a lexical category – is assigned to each word in a sentence
  
  \[ \text{walked: } S \backslash NP \text{ “give me an NP to my left and I return a sentence”} \]

- A small number of rules define how categories can combine
  
  - Rules based on the combinators from Combinatory Logic
CCG Lexical Categories

- Atomic categories: S, N, NP, PP, ... (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: S \NP walked
  - transitive verb: (S \NP )/NP respected
  - ditransitive verb: ((S \NP )/NP )/NP gave
- Complex categories can encode modification
  - PP nominal: (NP \NP )/NP
  - PP verbal: ((S \NP )(S \NP ))/NP
Simple CCG Derivation

\[
\begin{array}{c}
\text{interleukin} - 10 \quad \text{inhibits} \quad \text{production} \\
\hline
\text{NP} \quad (S\backslash NP)/NP \quad \text{NP} \\
\hline
S\backslash NP \quad S
\end{array}
\]

\[
> \quad \text{forward application} \\
< \quad \text{backward application}
\]
Function Application Schemata

- Forward (>) and backward (<) application:

\[
\begin{align*}
X/Y & \quad Y \quad \Rightarrow \quad X \quad (>) \\
Y & \quad X\backslash Y \quad \Rightarrow \quad X \quad (<)
\end{align*}
\]
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free

interleukin-10 \textit{inhibits} production
Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free

```
S
  /   \   
VP   NP
  /  \  /  
V   NP NP
  |   |  |
interleukin-10 inhibits production
```
Extraction out of a Relative Clause

The company which Microsoft bought

NP/N  N  (NP\NP)/(S/NP)  NP  (S\NP)/NP
Extraction out of a Relative Clause

The company which Microsoft bought

\[\text{NP}/N \quad N \quad (\text{NP}/\text{NP})/(\text{S}/\text{NP}) \quad \text{NP} \quad (\text{S}/\text{NP})/\text{NP} \]

\[\text{S}/(\text{S}/\text{NP}) \]

\[> T \quad \text{type-raising} \]
Extraction out of a Relative Clause

The company which Microsoft bought

NP/N  N  (NP\NP)/(S/NP)  NP  (S\NP)/NP

S/(S\NP)  S/NP

> T  type-raising
> B  forward composition
Extraction out of a Relative Clause

The company which Microsoft bought

NP/N  N  (NP/NP)/(S/NP)

NP  (S/NP)/NP

S/(S/NP) \rightarrow T

S/NP \rightarrow B

NP/NP
Extraction out of a Relative Clause

The company which Microsoft bought

NP/N N (NP\NP)/(S/NP) NP (S\NP)/NP

S/(S\NP) >T S/NP >B

NP \ NP

NP

Stephen Clark

Practical Linguistically Motivated Parsing

JHU, June 2009
Forward Composition and Type-Raising

- Forward composition ($\Rightarrow_B$):

  \[ X/Y \ Y/Z \Rightarrow X/Z \ (\Rightarrow_B) \]

- Type-raising ($T$):

  \[ X \Rightarrow T/(T\setminus X) \ (\Rightarrow_T) \]
  \[ X \Rightarrow T\setminus(T/X) \ (\Leftrightarrow_T) \]

- Extra combinatory rules increase the weak generative power to mild context-sensitivity
“Non-constituents” in CCG – Right Node Raising

Google \(\text{sells}\) but Microsoft \(\text{buys}\) shares

\[
\begin{align*}
  \frac{NP}{(S\backslash NP)/NP} & \quad \frac{NP}{(S\backslash NP)/NP} \\
  S/(S\backslash NP) & \quad \text{conj} & \quad (S\backslash NP)/NP & \quad NP
\end{align*}
\]

\(\triangleright \text{T} \quad \text{type-raising}\)
“Non-constituents” in CCG – Right Node Raising

Google sells but Microsoft buys shares

\[
\begin{align*}
\text{Google} & \quad \text{sells} & \quad \text{but} & \quad \text{Microsoft} & \quad \text{buys} & \quad \text{shares} \\
NP & \quad (S\backslash NP)/NP & \quad \text{conj} & \quad NP & \quad (S\backslash NP)/NP & \quad NP \\
S/(S\backslash NP) & \quad \Rightarrow^T & \quad S/\text{NP} & \quad \Rightarrow^T & \quad S/\text{NP} & \quad \Rightarrow^B \\
\end{align*}
\]

> T type-raising

> B forward composition
“Non-constituents” in CCG – Right Node Raising

Google sells but Microsoft buys shares

\[
\frac{NP}{S/(S\backslash NP)} \frac{(S\backslash NP)/NP}{NP} \quad \text{conj} \quad \frac{NP}{S/(S\backslash NP)} \frac{(S\backslash NP)/NP}{NP} \quad \frac{NP}{S/(S\backslash NP)}
\]

\[
\frac{S/\text{NP}}{S/\text{NP}} \quad >\text{T} \quad \frac{S/\text{NP}}{S/\text{NP}} \quad >\text{T} \quad \frac{S/\text{NP}}{S/\text{NP}} \quad >\text{B} \quad \frac{S/\text{NP}}{S/\text{NP}} \quad >\text{B} \quad \frac{S/\text{NP}}{S/\text{NP}} \quad <\Phi>
\]
“Non-constituents” in CCG – Right Node Raising

Google sells but Microsoft buys shares

\[
\begin{align*}
\text{NP} & \quad \text{NP} \\
\frac{S/(S\backslash NP)}{NP} & \quad \frac{(S\backslash NP)/NP}{NP}
\end{align*}
\]

\[
\begin{align*}
\frac{S/(S\backslash NP)}{S/ NP} & \quad \frac{S/(S\backslash NP)}{S/ NP}
\end{align*}
\]

\[
\begin{align*}
\frac{S/(S\backslash NP)}{S/ NP} & \quad \frac{S/(S\backslash NP)}{S/ NP}
\end{align*}
\]

Stephen Clark
Practical Linguistically Motivated Parsing
JHU, June 2009
Combinatory Categorial Grammar

- CCG is *mildly* context sensitive
- Natural language is provably non-context free
- Constructions in Dutch and Swiss German (Shieber, 1985) require more than context free power for their analysis
  - these have *crossing* dependencies (which CCG can handle)
CCG Semantics

- Categories encode argument sequences
- Parallel syntactic combinator operations and lambda calculus semantic operations

\[
\begin{align*}
John &
\vdash \text{NP} : \textit{john}' \\
shares &
\vdash \text{NP} : \textit{shares}' \\
\text{buys} &
\vdash (\text{S} \setminus \text{NP})/\text{NP} : \lambda x.\lambda y.\text{buys}'xy \\
\text{sleeps} &
\vdash \text{S} \setminus \text{NP} : \lambda x.\text{sleeps}'x \\
\text{well} &
\vdash (\text{S} \setminus \text{NP}) \setminus (\text{S} \setminus \text{NP}) : \lambda f.\lambda x.\text{well}'(fx)
\end{align*}
\]
## CCG Semantics

<table>
<thead>
<tr>
<th>Left arg.</th>
<th>Right arg.</th>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X/Y : f$</td>
<td>$Y : a$</td>
<td>Forward application</td>
<td>$X : f(a)$</td>
</tr>
<tr>
<td>$Y : a$</td>
<td>$X/Y : f$</td>
<td>Backward application</td>
<td>$X : f(a)$</td>
</tr>
<tr>
<td>$X/Y : f$</td>
<td>$Y/Z : g$</td>
<td>Forward composition</td>
<td>$X/Z : \lambda x.f(g(x))$</td>
</tr>
<tr>
<td>$X : a$</td>
<td></td>
<td>Type raising</td>
<td>$T/(T\setminus X) : \lambda f.f(a)$</td>
</tr>
</tbody>
</table>

etc.
Tree Adjoining Grammar
TAG Building Blocks

- Elementary trees (of many depths)
- Substitution at ↓
- Tree Substitution Grammar equivalent to CFG

\[ \alpha_3 \quad NP \quad \text{peanuts} \]

\[ \alpha_1 \quad NP \quad \text{Harry} \]

\[ \alpha_2 \quad S \]

\[ \text{NP} \quad \text{VP} \]

\[ \text{V} \quad \text{likes} \]

\[ \text{NP} \quad \]
TAG Building Blocks

- Auxiliary trees for *adjunction*
- Adds extra power beyond CFG

\[
\begin{align*}
\alpha_1 & \quad \text{NP} & \quad \text{Harry} \\
\alpha_2 & \quad \text{S} & \quad \text{NP} \downarrow & \quad \text{VP} \\
& & \quad \text{V} & \quad \text{NP} \downarrow & \quad \text{likes} \\
\alpha_3 & \quad \text{NP} & \quad \text{peanuts} \\
\beta & \quad \text{VP} & \quad \text{VP}^* & \quad \text{Adv} & \quad \text{passionately}
\end{align*}
\]
Harry likes peanuts passionately.

$\alpha_1$ likes

$\alpha_2$ Harry

$\beta$ passionately

$\alpha_3$ peanuts

$S$

NP Harry

VP$_1$

VP$_2$

V likes

NP peanuts

Adv passionately

Semantics

$Harry(x) \land likes(e, x, y) \land peanuts(y) \land passionately(e)$
**Why supertag?**

- If lexical items have more description associated with them, parsing is easier
  - Only useful if the supertag space is not huge

- Straightforward to compile parse from accurate supertagging
  - But impossible if there are any supertag errors
    - We can account for *some* supertag errors
    - Don’t always want a full parse anyway
What is Supertagging?

- Systematic assignment of supertags
- Supertags are:
  - Statistically selected
    - Robust
    - Tends to work
  - Linguistically motivated
    - This makes sense
**What is supertagging?**

- Many supertags for each word
  - **Extended Domain of Locality**
    - Each lexical item has one supertag for every syntactic environment it appears in.
    - Inspiration comes from LTAG, lexicalized tree-adjoining grammars, in which *all* dependencies are localized.
  - Generally, agreement features such as number and tense, are not part of the supertag.
HOW TO SUPERTAG

“How Alice opened her eyes and saw.”

Supertags:

- Verb
  - Transitive verb
  - Intransitive verb
  - Infinitive verb
  - ...

- Noun
  - Noun phrase (subject)
  - Nominal predicative
  - Nominal modifier
  - Nominal predicative subject extraction
  - ...
“Alice opened her eyes and saw.”

Supertags:
- Verb
  - Transitive verb
  - Intransitive verb
  - Infinitive verb
  - ...
- Noun
  - Noun phrase (subject)
  - Nominal predicative
  - Nominal modifier
  - Nominal predicative subject extraction
  - ...

Diagram: S \(\rightarrow\) VP \(\rightarrow\) NP \(\downarrow\) saw \(\rightarrow\) NP \(\downarrow\)
HOW TO SUPERTAG

- A supertag can be ruled out for a given word in a given input string...
  - Left and/or right context is too long/short for the input
  - If the supertag contains other terminals not found in the input
HOW TO SUPERTAG

“Alice opened her eyes and saw.”

Supertags:

- Verb
  - Transitive verb
  - Intransitive verb
  - Infinitive verb
  - ...
- Noun
  - Noun phrase (subject)
  - Nominal predicative
  - Nominal modifier
  - Nominal predicative subject extraction
  - ...

... to saw ...

[Diagram of sentence structure with terms indicated as nodes and edges]
HOW TO SUPERTAG

- This works fairly well
  - 50% average reduction in number of possible supertags
HOW TO SUPERTAG

...but there's more to be done

- Good: average number of possible supertags per word reduced from 47 to 25
- Bad: average of 25 possible supertags per word
HOW TO SUPERTAG

- Disambiguation by unigrams?
  - Give each word its most frequent supertag after PoS tagging
    - ~75% accurate
      - Better results than one might expect given large number of possible supertags
      - Common words (determiners, etc.) usually correct
        - This helps accuracy
    - Back off to PoS for unknown words
      - Also usually correct
**How to Supertag**

- Disambiguation by n-grams?

  \[
  T = \arg \max_{T} \Pr(T_1, T_2, ..., T_N) \times \Pr(W_1, W_2, ..., W_N|T_1, T_2, ..., T_N)
  \]

  - We assume that subsequent words are independent

    \[
    \Pr(W_1, W_2, ..., W_N|T_1, T_2, ..., T_N) \approx \prod_{i=1}^{N} \Pr(W_i|T_i)
    \]

  - Trigrams plus Good-Turing smoothing
    - Accuracy around 90%
      - Versus 75% from unigrams
    - Contextual information more important than lexical
      - Reversal of trend for PoS tagging
However...

- Correctly supertagged text yields a 30X parsing speedup
  - But even one mistake can cause parsing to fail completely
    - This is rather likely

- Solution: n-best supertags?
  - When n=3, we get up to 96% accuracy...
    - Not bad at all for such a simple method
    - 425 lexical categories (PTB-CFG: ~50)
    - 12 combinatory rules (PTB-CFG: > 500,000)
Adaptive Supertagging [Clark & Curran, 2007]

Start with an initial prob. cutoff $\beta$

<table>
<thead>
<tr>
<th>He</th>
<th>reads</th>
<th>the book</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP$</td>
<td>$(S[pss]\backslash NP)/NP$</td>
<td>$NP/N$</td>
</tr>
</tbody>
</table>
Adaptive Supertagging [Clark & Curran, 2007]

Prune a category, if its probability is below $\beta$ times the prob. of the best category

<table>
<thead>
<tr>
<th>He</th>
<th>reads</th>
<th>the book</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(S[pss]\NP)/NP</td>
<td>NP/N</td>
</tr>
</tbody>
</table>
Adaptive Supertagging \textbf{[Clark & Curran, 2007]}

Decrease $\beta$ if no spanning analysis

<table>
<thead>
<tr>
<th>He</th>
<th>reads</th>
<th>the</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>$(S[pss]\backslash NP)/NP$</td>
<td>NP/N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>$(S\backslash NP)/NP$</td>
<td>NP/N</td>
<td>$(S\backslash NP)/NP$</td>
</tr>
<tr>
<td>N/N</td>
<td>$S\backslash NP$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decrease $\beta$ if no spanning analysis

He

reads

the

book

\[ \begin{array}{cccc}
\text{NP} & (S[pss]\backslash \text{NP})/\text{NP} & \text{NP}/\text{N} & \text{N} \\
N & (S\backslash \text{NP})/\text{NP} & \text{NP}/\text{NP} & (S\backslash \text{NP})/\text{NP} \\
N/\text{N} & S\backslash \text{NP} & \text{NP}/\text{NP} & \text{NP}/\text{NP} \\
\text{NP}/\text{NP} & (S[pt]\backslash \text{NP})/\text{NP} & \text{NP}/\text{NP} & \text{NP}/\text{NP} \\
\end{array} \]
Recurrent neural networks (RNN)
Recurrent neural networks

- Use the same computational function and parameters across different time steps of the sequence
- Each time step: takes the input entry and the previous hidden state to compute the output entry
- Loss: typically computed every time step
Figure from *Deep Learning*, by Goodfellow, Bengio and Courville
Recurrent neural networks

Math formula:

\[ a^{(t)} = b + W s^{(t-1)} + U x^{(t)} \]
\[ s^{(t)} = \tanh(a^{(t)}) \]
\[ o^{(t)} = c + V s^{(t)} \]
\[ \hat{y}^{(t)} = \text{softmax}(o^{(t)}) \]

Figure from Deep Learning, Goodfellow, Bengio and Courville
Advantage

• Hidden state: a lossy summary of the past
• Shared functions and parameters: greatly reduce the capacity and good for generalization in learning
• Explicitly use the prior knowledge that the sequential data can be processed by in the same way at different time step (e.g., NLP)
Advantage

• Hidden state: a lossy summary of the past
• Shared functions and parameters: greatly reduce the capacity and good for generalization in learning
• Explicitly use the prior knowledge that the sequential data can be processed by in the same way at different time step (e.g., NLP)

• Yet still powerful (actually universal): any function computable by a Turing machine can be computed by such a recurrent network of a finite size (see, e.g., Siegelmann and Sontag (1995))
Figure 4: A simple recurrent network.
Recurrent Network Variations

- This network can theoretically learn contexts arbitrarily far back
- Many structural variations
  - Elman/Simple Net
  - Jordan Net
  - Mixed
  - Context sub-blocks, etc.
  - Multiple hidden/context layers, etc.
  - Generalized row representation
- How do we learn the weights?
Simple Recurrent Training – Elman Training

- Can think of net as just being a normal MLP structure where part of the input happens to be a copy of the last set of state/hidden node activations. The MLP itself does not even need to be aware that the context inputs are coming from the hidden layer.
- Then can train with standard BP training.
- While network can theoretically look back arbitrarily far in time, Elman learning gradient goes back only 1 step in time, thus limited in the context it can learn.
  - Would if current output depended on input 2 time steps back.
- Can still be useful for applications with short term dependencies.
BPTT – Backprop Through Time

- BPTT allows us to look back further as we train
- However we have to pre-specify a value $k$, which is the maximum that learning will look back
- During training we *unfold* the network in time as if it were a standard feedforward network with $k$ layers
  - But where the weights of each unfolded layer are the same (shared)
- We then train the unfolded $k$ layer feedforward net with standard BP
- Execution still happens with the actual recurrent version
- Is not knowing $k$ apriori that bad? How do you choose it?
  - Cross Validation, just like finding best number of hidden nodes, etc., thus we can find a good $k$ fairly reasonably for a given task
  - But problematic if the amount of state needed varies a lot
• $k$ is the number of feedback/context blocks in the unfolded net.
• Note $k=1$ is just standard MLP with no feedback
• 1st block $h(0)$ activations are just initialized to a constant or 0 so $k=1$ is still same as standard MLP, so just leave it out for feedforward MLP
• Last context block is $h(k-1)$
• $k=2$ is Elman training
Training RNN

• Principle: unfold the computational graph, and use backpropagation
• Called back-propagation through time (BPTT) algorithm
• Can then apply any general-purpose gradient-based techniques
Training RNN

• Principle: unfold the computational graph, and use backpropagation
• Called back-propagation through time (BPTT) algorithm
• Can then apply any general-purpose gradient-based techniques

• Conceptually: first compute the gradients of the internal nodes, then compute the gradients of the parameters
Recurrent neural networks

Math formula:

\[ a^{(t)} = b + W s^{(t-1)} + U x^{(t)} \]
\[ s^{(t)} = \tanh(a^{(t)}) \]
\[ o^{(t)} = c + V s^{(t)} \]
\[ \hat{y}^{(t)} = \text{softmax}(o^{(t)}) \]
Gradient at $L^{(t)}$: (total loss is sum of those at different time steps)

$$\frac{\partial L}{\partial L^{(t)}} = 1.$$
Gradient at $o^{(t)}$:

$$\frac{\partial L}{\partial o^{(t)}_i} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o^{(t)}_i} = \hat{y}^{(t)}_k - 1_{i,y^{(t)}}$$
Gradient at $s^{(\tau)}$:

$$\left( \nabla_{o^{(\tau)}} L \right) \frac{\partial O^{(\tau)}}{\partial S^{(\tau)}} = \left( \nabla_{o^{(\tau)}} L \right) V$$
Gradient at $s^{(t)}$:

$$\left( \nabla_{s^{(t+1)}} L \right) \frac{\partial s^{(t+1)}}{\partial s^{(t)}} + \left( \nabla_{o^{(t)}} L \right) \frac{\partial o^{(t)}}{\partial s^{(t)}}$$

Figure from *Deep Learning*, Goodfellow, Bengio and Courville
Gradient at parameter $V$:

$$
\sum_t (\nabla_{o(t)} L) \frac{\partial o(t)}{\partial V} = \sum_t (\nabla_{o(t)} L) s^{(t)^T}
$$
Supertagging with a RNN

• Using only dense features
  – word embedding
  – suffix embedding
  – capitalization

• The input layer is a concatenation of all embeddings of all words in a context window
Supertagging with a RNN

... bought some books and ...

...
Supertagging with a RNN

... bought some books and ...

...
Supertagging with a RNN

... bought some books and ...

...
Supertagging with a RNN

... bought some books and ...
Supertagging with a RNN

... bought some books and ...
### 1-best Supertagging Results: dev

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C&amp;C (gold POS)</td>
<td>92.60</td>
<td>-</td>
</tr>
<tr>
<td>C&amp;C (auto POS)</td>
<td>91.50</td>
<td>0.57</td>
</tr>
<tr>
<td>NN</td>
<td>91.10</td>
<td>21.00</td>
</tr>
<tr>
<td>RNN</td>
<td>92.63</td>
<td>-</td>
</tr>
<tr>
<td>RNN+dropout</td>
<td>93.07</td>
<td>2.02</td>
</tr>
</tbody>
</table>

Table 1: 1-best tagging accuracy and speed comparison on CCGBank Section 00 with a single CPU core (1,913 sentences), tagging time in secs.
## 1-best Supertagging Results: test

<table>
<thead>
<tr>
<th>Model</th>
<th>Section 23</th>
<th>Wiki</th>
<th>Bio</th>
</tr>
</thead>
<tbody>
<tr>
<td>C&amp;C (gold POS)</td>
<td>93.32</td>
<td>88.80</td>
<td>91.85</td>
</tr>
<tr>
<td>C&amp;C (auto POS)</td>
<td>92.02</td>
<td>88.80</td>
<td>89.08</td>
</tr>
<tr>
<td>NN</td>
<td>91.57</td>
<td>89.00</td>
<td>88.16</td>
</tr>
<tr>
<td>RNN</td>
<td>93.00</td>
<td>90.00</td>
<td>88.27</td>
</tr>
</tbody>
</table>

Table 2: 1-best tagging accuracy comparison on CCGBank Section 23 (2,407 sentences), Wikipedia (200 sentences) and Bio-GENIA (1,000 sentences).
Multi-tagging Results: dev

![Graph showing multi-tagging accuracy vs ambiguity level with different models: RNN + dropout, RNN, NN, C&C. The graph demonstrates the performance of each model across various ambiguity levels, with RNN + dropout showing the highest accuracy.]
Multi-tagging Results: test

![Graph showing multi-tagging accuracy vs ambiguity level]

- **Multi-tagging accuracy**
- **Ambiguity level**
- **RNN + dropout**
- **NN**
- **C&C**
Final Parsing Results

<table>
<thead>
<tr>
<th></th>
<th>CCGBank Section 23</th>
<th>Wikipedia</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
</tr>
<tr>
<td>C&amp;C</td>
<td>86.24</td>
<td>84.85</td>
</tr>
<tr>
<td>(NN)</td>
<td>86.71</td>
<td>85.56</td>
</tr>
<tr>
<td>(RNN)</td>
<td><strong>87.68</strong></td>
<td><strong>86.47</strong></td>
</tr>
</tbody>
</table>

Table 3: Parsing test results (auto POS). We evaluate on all sentences (100% coverage) as well as on only those sentences that returned spanning analyses (% cov.). RNN and NN both have 100% coverage on the Wikipedia data.