4) Categorial Grammars

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge
Reminder:

For statistical parsing we need:

- a grammar,
- a parsing algorithm,
- a scoring model for parses,
- an algorithm for finding best parse.

- Parsing **efficiency** is dependent on the parsing and best-parse algorithms.
- Parsing **accuracy** is dependent on the grammar and scoring model.

- Often there is a trade-off between using a more sophisticated (and perhaps less robust) grammar formalism at the expense of efficiency.
Reminder:

For statistical parsing we need:

- a grammar,
- a parsing algorithm,
- a scoring model for parses,
- an algorithm for finding best parse.

Parsing **efficiency** is dependent on the parsing and best-parse algorithms.

Parsing **accuracy** is dependent on the grammar and scoring model.

Often there is a trade-off between using a more sophisticated (and perhaps less robust) grammar formalism at the expense of efficiency.
For statistical parsing we need:

- a grammar,
- a parsing algorithm,
- a scoring model for parses,
- an algorithm for finding best parse.

Parsing **efficiency** is dependent on the parsing and best-parse algorithms.

Parsing **accuracy** is dependent on the grammar and scoring model.

Often there is a trade-off between using a more sophisticated (and perhaps less robust) grammar formalism at the expense of efficiency.
CCG parsers exist that are robust and efficient

- **Combinatory Categorial Grammars** parsers exist that are robust and efficient (Clark & Currans 2007)
  https://github.com/chrzyki/candc
- CCGs provide a mapping between syntactic structure and predicate-argument structure

- The C&C parser uses a discriminative model over complete parses
- A **supertagging** phase is needed before parsing commences

First, reminder of a CCG?
CCG parsers exist that are robust and efficient

- **Combinatory Categorial Grammars** parsers exist that are robust and efficient (Clark & Currans 2007)
  https://github.com/chrzyki/candc
- CCGs provide a mapping between syntactic structure and predicate-argument structure
  - The C&C parser uses a discriminative model over complete parses
  - A **supertagging** phase is needed before parsing commences

First, reminder of a CCG?
Categorial grammars are **lexicalized grammars**

In a **classic categorial grammar** each symbol in the alphabet is associated with a finite number of **types**.

- Types are formed from primitive types using two operators, \ and /.
- If \( P_r \) is the set of **primitive types** then the set of all types, \( T_p \), satisfies:
  - \( P_r \subset T_p \)
  - if \( A \in T_p \) and \( B \in T_p \) then \( A \setminus B \in T_p \)
  - if \( A \in T_p \) and \( B \in T_p \) then \( A / B \in T_p \)

- Note that it is possible to arrange types in a hierarchy: a type \( A \) is a **subtype** of \( B \) if \( A \) occurs in \( B \) (that is, \( A \) is a subtype of \( B \) iff \( A = B \); or \( B = B_1 \setminus B_2 \) or \( B = B_1 / B_2 \)) and \( A \) is a subtype of \( B_1 \) or \( B_2 \).
Categorial grammars are **lexicalized grammars**

- A relation, $\mathcal{R}$, maps symbols in the alphabet $\Sigma$ to members of $T_p$.
- A grammar that associates at most one type to each symbol in $\Sigma$ is called a **rigid grammar**.
- A grammar that assigns at most $k$ types to any symbol is a **$k$-valued grammar**.
- We can define a classic categorial grammar as $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:
  - $\Sigma$ is the alphabet/set of terminals
  - $P_r$ is the set of primitive types
  - $S$ is a distinguished member of the primitive types $S \in P_r$ that will be the root of complete derivations
  - $\mathcal{R}$ is a relation $\Sigma \times T_p$ where $T_p$ is the set of all types as generated from $P_r$ as described above
Categorial grammars are *lexicalized grammars*

A string has a valid parse if the types assigned to its symbols can be combined to produce a derivation tree with root $S$.

Types may be combined using the two rules of **function application**:

- **Forward application** is indicated by the symbol $>$:
  \[
  \frac{A/B \quad B}{A} >
  \]

- **Backward application** is indicated by the symbol $<$:
  \[
  \frac{B \quad A\backslash B}{A} <
  \]
Categorial grammars are **lexicalized grammars**

Derivation tree for the string *xyz* using the grammar $G_{cg} = (\Sigma, Pr, S, \mathcal{R})$ where:

- $Pr = \{S, A, B\}$
- $\Sigma = \{x, y, z\}$
- $S = S$
- $\mathcal{R} = \{(x, A), (y, S\backslash A/B), (z, B)\}$
Categorial grammars are **lexicalized grammars**

Derivation tree for the string *Alice chases rabbits* using the grammar $G_{cg} = (\Sigma, P_r, S, \mathcal{R})$ where:

$P_r = \{S, NP\}$  
$
\Sigma = \{alice, chases, rabbits\}$  
$S = S$  
$\mathcal{R} = \{(alice, NP), (chases, S\backslash NP / NP), (rabbits, NP)\}$

$$
\frac{alice}{NP} \quad \frac{chases}{S\backslash NP/\NP} \quad \frac{rabbits}{\NP} \\
\mathcal{R} \quad \mathcal{R} \quad \mathcal{R}
$$

$S \ (<) \ S\backslash NP \ (>)$

NP

alice
chases
rabbits

NP

S\backslash NP / NP

S\backslash NP

S

NP

S\backslash NP/\NP

NP
Combination categorial grammars extend classic CG

Combination categorial grammars use function composition rules in addition to function application:

- **Forward composition** is indicated by the symbol $> B$:
  \[
  \frac{X/Y \quad Y/Z}{X/Z} > B
  \]

- **Backward composition** is indicated by the symbol $< B$:
  \[
  \frac{Y \backslash Z \quad X \backslash Y}{X \backslash Z} < B
  \]

They also use **type-raising** rules (only applies to $NP$, $PP$, $S[adj] \backslash NP$):

\[
\frac{X}{T/(T \backslash X)} \quad T
\]
\[
\frac{X}{T \backslash (T/X)} \quad T
\]

- Also backward crossed composition and co-ordination (see Steedman)
CCG examples...
Lexicalised grammar parsers have three steps

Parsing with lexicalised grammar formalisms has the following pipeline:

1. Lexical **categories are assigned** to each word in the sentence
2. Parser combines the categories together to **form legal structures**
3. The **highest scoring derivation** is found according to some model

For C&C:

1. Uses a **supertagger** (log-linear model using words and PoS tags in a 5-word window)
2. Uses the CKY chart parsing algorithm to derive all legal structures
3. Uses Viterbi to find best parse (log-linear model to score parses based on their features)

- **CCGBank derived from the Penn Treebank is used to train the scoring models**
Lexicalised grammar parsers have three steps

Parsing with lexicalised grammar formalisms has the following pipeline:

1. Lexical categories are assigned to each word in the sentence
2. Parser combines the categories together to form legal structures
3. The highest scoring derivation is found according to some model

For C&C:

1. Uses a supertagger (log-linear model using words and PoS tags in a 5-word window)
2. Uses the CKY chart parsing algorithm to derive all legal structures
3. Uses Viterbi to find best parse (log-linear model to score parses based on their features)

- *CCGBank derived from the Penn Treebank is used to train the scoring models*
The C&C parser uses a supertagger

Two stage tagging using **log-linear model**:

\[
P(tag|context) = \frac{1}{Z} \exp \sum_i \lambda_i f_i(tag, context)
\]

where \( \lambda_i \) is the weight of the \( i \)th feature, \( f_i \) (and \( Z \) is a normalising factor)

- context is **5-word window** surrounding target word
- features are
  - words and POS tags in the context window
  - two previously assigned categories
- \( \approx 400 \) lexical categories
- baseline for task is \( \approx 72\% \) (compare to normal POS tagging \( \approx 90\% \))
- One tag per word yields \( \approx 92\% \) — improve by assigning all categories whose probability is within some factor \( \beta \) of the highest probability category
The C&C parser uses CKY and Viterbi

- Build a packed chart of all the trees using CKY.
- Parses are scored according to their features (feature forest)
  - Discriminative parser: $P(\text{tree}|\text{words}) = \frac{1}{Z_W} \exp^{\lambda.F(\text{tree})}$
    - where $\lambda.F(\text{tree}) = \sum_i \lambda_i f_i(\text{tree})$ and $\lambda_i$ is the weight of the $i$th feature, $f_i$ (and $Z_W$ is a normalising factor)
  - Train $\lambda$ by maximising log-likelihood over the training data (minus a prior term to prevent overfitting)
- Use CKY and Viterbi when decoding to find the best parse.
- Packing requires that any rule based features are local—confined to a single rule application.
The C&C parser uses CKY and Viterbi

- Build a packed chart of all the trees using CKY.
- Parses are scored according to their features (feature forest)
- Discriminative parser: \( P(\text{tree}|\text{words}) = \frac{1}{Z_W} \exp^{\lambda.F(\text{tree})} \)
  where \( \lambda.F(\text{tree}) = \sum_i \lambda_i f_i(\text{tree}) \) and \( \lambda_i \) is the weight of the \( i \)th feature, \( f_i \) (and \( Z_W \) is a normalising factor)
- Train \( \lambda \) by maximising log-likelihood over the training data (minus a prior term to prevent overfitting)
- Use CKY and Viterbi when decoding to find the best parse.
- Packing requires that any rule based features are local—confined to a single rule application.
The C&C parser uses CKY and Viterbi

- Build a packed chart of all the trees using CKY.
- Parses are scored according to their features (feature forest)
- Discriminative parser: \( P(tree|words) = \frac{1}{Z_W} \exp^{\lambda.F(tree)} \)
  where \( \lambda.F(tree) = \sum_i \lambda_i f_i(tree) \) and \( \lambda_i \) is the weight of the \( i \)th feature, \( f_i \) (and \( Z_W \) is a normalising factor)
- Train \( \lambda \) by maximising log-likelihood over the training data (minus a prior term to prevent overfitting)
- Use CKY and Viterbi when decoding to find the best parse.
- Packing requires that any rule based features are local—confined to a single rule application.
The features used in the C&C scoring model include:

- features encoding local trees (that is two combining categories and the result category)
- features encoding word-lexical category pairs at the leaves of the derivation
- features encoding grammatical dependencies, including the distance between them
CKY CCG parse example...