Overview

- Log-linear models for parsing and other problems
- Global Linear Models

A General Approach: (Conditional) History-Based Models

- We’ve shown how to define $P(T \mid S)$ where $T$ is a tag sequence
- How do we define $P(T \mid S)$ if $T$ is a parse tree (or another structure)?

Step 1: represent a tree as a sequence of decisions $d_1 \ldots d_m$

$$T = \langle d_1, d_2, \ldots d_m \rangle$$

$m$ is not necessarily the length of the sentence

Step 2: the probability of a tree is

$$P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S)$$

Step 3: Use a log-linear model to estimate

$$P(d_i \mid d_1 \ldots d_{i-1}, S)$$

Step 4: Search?? (answer we’ll get to later: beam or heuristic search)
Ratnaparkhi’s Parser: Three Layers of Structure

1. Part-of-speech tags
2. Chunks
3. Remaining structure

Layer 1: Part-of-Speech Tags

- Step 1: represent a tree as a sequence of decisions $d_1 \ldots d_m$
  
  $T = \langle d_1, d_2, \ldots d_m \rangle$

- First $n$ decisions are tagging decisions
  
  $\langle d_1 \ldots d_n \rangle = \langle DT, NN, Vt, DT, NN, IN, DT, NN \rangle$

Layer 2: Chunks

Chunks are defined as any phrase where all children are part-of-speech tags

(Other common chunks are ADJP, QP)
Layer 2: Chunks

- Step 1: represent a tree as a sequence of decisions $d_1 \ldots d_n$
  
  $$T = \langle d_1, d_2, \ldots, d_n \rangle$$

- First $n$ decisions are tagging decisions
  Next $n$ decisions are chunk tagging decisions
  
  $$\langle d_1 \ldots d_{2n} \rangle = \langle \text{DT, NN, Vt, DT, NN, IN, DT, NN, Start(NP), Join(NP), Other, Start(NP), Join(NP), Other, Start(NP), Join(NP)} \rangle$$

Layer 3: Remaining Structure

Alternate Between Two Classes of Actions:

- Join(X) or Start(X), where X is a label (NP, S, VP etc.)

- Check=YES or Check=NO

Meaning of these actions:

- Start(X) starts a new constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Join(X) continues a constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Check=NO does nothing
- Check=YES takes previous Join or Start action, and converts it into a completed constituent
The lawyer questioned the witness about the revolver.

Check=NO
The lawyer questioned the witness about the revolver.

Check=NO
the lawyer questioned the witness about the revolver.
The Final Sequence of decisions

\[ \langle d_1 \ldots d_m \rangle = \langle \text{DT, NN, Vt, DT, NN, IN, DT, NN,} \]
\[ \text{Start(NP), Join(NP), Other, Start(NP), Join(NP),} \]
\[ \text{Other, Start(NP), Join(NP),} \]
\[ \text{Start(S), Check=NO, Start(VP), Check=NO,} \]
\[ \text{Join(VP), Check=NO, Start(PP), Check=NO,} \]
\[ \text{Join(PP), Check=NO,} \]
\[ \text{Check=YES, Join(VP), Check=YES,} \]
\[ \text{Join(S), Check=YES} \rangle \]

A General Approach: (Conditional) History-Based Models

- Step 1: represent a tree as a sequence of decisions \( d_1 \ldots d_m \)
  \[ T = \langle d_1, d_2, \ldots d_m \rangle \]
  \( m \) is not necessarily the length of the sentence

- Step 2: the probability of a tree is
  \[ P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S) \]

- Step 3: Use a log-linear model to estimate
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) \]

- Step 4: Search?? (answer we’ll get to later: beam or heuristic search)

Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) \]

- A reminder:
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\phi(\langle d_1 \ldots d_{i-1}, S \rangle, d_i) \cdot W}}{\sum_{d \in \mathcal{A}} e^{\phi(\langle d_1 \ldots d_{i-1}, S \rangle, d) \cdot W}} \]
  where:
  \( \langle d_1 \ldots d_{i-1}, S \rangle \) is the history
  \( d_i \) is the outcome
  \( \phi \) maps a history/outcome pair to a feature vector
  \( W \) is a parameter vector
  \( \mathcal{A} \) is set of possible actions
  (may be context dependent)
Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\phi(d_1 \ldots d_{i-1}, S, d_i) \cdot w}}{\sum_{d \in A} e^{\phi(d_1 \ldots d_{i-1}, S, d) \cdot w}} \]

- The big question: how do we define \( \phi \)?
- Ratnaparkhi’s method defines \( \phi \) differently depending on whether next decision is:
  - A tagging decision
    (same features as before for POS tagging!)
  - A chunking decision
  - A start/join decision after chunking
  - A check=no/check=yes decision

Layer 2: Chunks

<table>
<thead>
<tr>
<th>Start(NP)</th>
<th>Join(NP)</th>
<th>Other</th>
<th>Start(NP)</th>
<th>Join(NP)</th>
<th>IN</th>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT</td>
<td>NN</td>
<td>Vt</td>
<td>DT</td>
<td>NN</td>
<td>about</td>
<td>the</td>
<td>the</td>
</tr>
<tr>
<td>the</td>
<td>lawyer</td>
<td>questioned</td>
<td>the</td>
<td>witness</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ “TAG=Join(NP);Word0=witness;POS0=NN”
  “TAG=Join(NP);POS0=NN”
  “TAG=Join(NP);Word+1=about;POS+1=IN”
  “TAG=Join(NP);POS+1=IN”
  “TAG=Join(NP);Word+2=the;POS+2=DT”
  “TAG=Join(NP);POS+2=IN”
  “TAG=Join(NP);Word-1=the;POS-1=DT;TAG-1=Start(NP)”
  “TAG=Join(NP);POS-1=DT;TAG-1=Start(NP)”
  “TAG=Join(NP);TAG-1=Start(NP)”

Layer 3: Join or Start

- Looks at head word, constituent (or POS) label, and start/join annotation of \( n \)’th tree relative to the decision, where \( n = -2, -1 \)
- Looks at head word, constituent (or POS) label of \( n \)’th tree relative to the decision, where \( n = 0, 1, 2 \)
- Looks at bigram features of the above for (-1,0) and (0,1)
- Looks at trigram features of the above for (-2,-1,0), (-1,0,1) and (0, 1, 2)
- The above features with all combinations of head words excluded
- Various punctuation features

Layer 3: Check=NO or Check=YES

- A variety of questions concerning the proposed constituent
The Search Problem

- In POS tagging, we could use the Viterbi algorithm because
  \[ P(t_j \mid w_1 \ldots w_n, j, t_1 \ldots t_{j-1}) = P(t_j \mid w_1 \ldots w_n, j, t_{j-2} \ldots t_{j-1}) \]

- Now: Decision \( d_i \) could depend on arbitrary decisions in the “past” \( \Rightarrow \) no chance for dynamic programming

- Instead, Ratnaparkhi uses a beam search method

Global Linear Models: Overview

- A brief review of history-based methods

- A new framework: Global linear models

- Parsing problems in this framework:
  - Reranking problems

- Parameter estimation method 1:
  - A variant of the perceptron algorithm

Techniques

- So far:
  - Smoothed estimation
  - Probabilistic context-free grammars
  - Log-linear models
  - Hidden markov models
  - History-based models

- Today:
  - Global linear models

Supervised Learning in Natural Language

- General task: induce a function \( F \) from members of a set \( \mathcal{X} \) to members of a set \( \mathcal{Y} \). e.g.,

<table>
<thead>
<tr>
<th>Problem</th>
<th>( x \in \mathcal{X} )</th>
<th>( y \in \mathcal{Y} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>sentence</td>
<td>parse tree</td>
</tr>
<tr>
<td>Machine translation</td>
<td>French sentence</td>
<td>English sentence</td>
</tr>
<tr>
<td>POS tagging</td>
<td>sentence</td>
<td>sequence of tags</td>
</tr>
</tbody>
</table>

- Supervised learning:
  - we have a training set \( (x_i, y_i) \) for \( i = 1 \ldots n \)
The Models so far

- Most of the models we’ve seen so far are **history-based models**:
  - We break structures down into a derivation, or sequence of decisions
  - Each decision has an associated conditional probability
  - Probability of a structure is a product of decision probabilities
  - Parameter values are estimated using variants of maximum-likelihood estimation
  - Function $F: \mathcal{X} \rightarrow \mathcal{Y}$ is defined as
    $$ F(x) = \arg\max_y P(y, x | \Theta) $$
    or
    $$ F(x) = \arg\max_y P(y | x, \Theta) $$

---

**Example 1: PCFGs**

- We break structures down into a derivation, or sequence of decisions
  - We have a top-down derivation, where each decision is to expand some non-terminal $\alpha$ with a rule $\alpha \rightarrow \beta$
  - Each decision has an associated conditional probability $\alpha \rightarrow \beta$ has probability $P(\alpha \rightarrow \beta | \alpha)$
  - Probability of a structure is a product of decision probabilities
    $$ P(T, S) = \prod_{i=1}^{n} P(\alpha_i \rightarrow \beta_i | \alpha_i) $$
    where $\alpha_i \rightarrow \beta_i$ for $i = 1 \ldots n$ are the $n$ rules in the tree
  - Parameter values are estimated using variants of maximum-likelihood estimation
    $$ P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)} $$
  - Function $F: \mathcal{X} \rightarrow \mathcal{Y}$ is defined as
    $$ F(x) = \arg\max_y P(y, x | \Theta) $$
    Can be computed using dynamic programming

---

**Example 2: Log-linear Taggers**

- We break structures down into a derivation, or sequence of decisions
  - For a sentence of length $n$ we have $n$ tagging decisions, in left-to-right order
  - Each decision has an associated conditional probability
    $$ P(t_i | t_{i-1}, t_{i-2}, w_1 \ldots w_n) $$
    where $t_i$ is the $i$'th tagging decision, $w_i$ is the $i$'th word
  - Probability of a structure is a product of decision probabilities
    $$ P(t_1 \ldots t_n | w_1 \ldots w_n) = \prod_{i=1}^{n} P(t_i | t_{i-1}, t_{i-2}, w_1 \ldots w_n) $$
  - Parameter values are estimated using variants of maximum-likelihood estimation
    $$ P(t_i | t_{i-1}, t_{i-2}, w_1 \ldots w_n) $$
    is estimated using a log-linear model
  - Function $F: \mathcal{X} \rightarrow \mathcal{Y}$ is defined as
    $$ F(x) = \arg\max_y P(y | x, \Theta) $$
    Can be computed using dynamic programming

---

**A New Set of Techniques: Global Linear Models**

Overview of today’s lecture:

- **Global linear models** as a framework
- Parsing problems in this framework:
  - Reranking problems
- A variant of the perceptron algorithm
Global Linear Models as a Framework

- We’ll move away from history-based models
  No idea of a ‘derivation’, or attaching probabilities to ‘decisions’
- Instead, we’ll have feature vectors over entire structures
  “Global features”
- First piece of motivation:
  Freedom in defining features

An Example: Parsing

- In lecture 4, we described lexicalized models for parsing
- Showed how a rule can be generated in a number of steps

Model 2

- Step 1: generate category of head child
  
  \[ P_h(VP \mid S, \text{told, } V[6]) \]

- Step 2: choose left subcategorization frame
  
  \[ P_h(VP \mid S, \text{told, } V[6]) \times P_{lc}(\{NP-C\} \mid S, VP, \text{told, } V[6]) \]
• Step 3: generate left modifiers in a Markov chain

\[ S(told,V[6]) \]
\[ \downarrow \]
\[ S(told,V[6]) \]
\[ NP-C(Hillary,NNP) \]
\[ VP(told,V[6]) \]
\[ \{NP-C\} \]

\[ P_h(VP \mid S, told, V[6]) \times P_{lc}(\{NP-C\} \mid S, VP, told, V[6]) \times \]
\[ P_d(NP-C(Hillary,NNP) \mid S,VP,told,V[6],LEFT,\{NP-C\}) \times \]
\[ P_d(NP(yesterday,NN) \mid S,VP,told,V[6],LEFT,\{\}) \times \]
\[ P_d(STOP \mid S,VP,told,V[6],LEFT,\{\}) \]

The Probabilities for One Rule

\[ S(told,V[6]) \]
\[ \downarrow \]
\[ S(told,V[6]) \]
\[ NP(yesterday,NN) \]
\[ NP-C(Hillary,NNP) \]
\[ VP(told,V[6]) \]
\[ \{\} \]

\[ P_h(VP \mid S, told, V[6]) \times \]
\[ P_{lc}(\{NP-C\} \mid S, VP, told, V[6]) \times \]
\[ P_d(NP-C(Hillary,NNP) \mid S,VP,told,V[6],LEFT,\{NP-C\}) \times \]
\[ P_d(NP(yesterday,NN) \mid S,VP,told,V[6],LEFT,\{\}) \times \]
\[ P_d(STOP \mid S,VP,told,V[6],RIGHT,\{\}) \times \]

Three parameter types:
Head parameters, Subcategorization parameters, Dependency parameters
**Smoothed Estimation**

\[ P(NP(\_, NN) \ VP \mid S(\text{questioned}, Vt)) = \]

\[
\lambda_1 \times \frac{\text{Count}(S(\text{questioned}, Vt) \rightarrow NP(\_, NN) \ VP)}{\text{Count}(S(\text{questioned}, Vt))} \\
+ \lambda_2 \times \frac{\text{Count}(S(\_, Vt) \rightarrow NP(\_, NN) \ VP)}{\text{Count}(S(\_, Vt))}
\]

- Where \( 0 \leq \lambda_1, \lambda_2 \leq 1 \), and \( \lambda_1 + \lambda_2 = 1 \)

**An Example: Parsing**

- In lecture 4, we described lexicalized models for parsing
  - Showed how a rule can be generated in a number of steps
  - The end result:
    - We have head, dependency, and subcategorization parameters
    - Smoothed estimation means “probability” of a tree depends on counts of many different types of events
  - What if we want to add new features? Can be very awkward to incorporate some features in history-based models

**A Need for Flexible Features**

Example 1: Parallelism in coordination [Johnson et. al 1999]

Constituents with similar structure tend to be coordinated
⇒ how do we allow the parser to learn this preference?

Bars in New York and pubs in London vs. Bars in New York and pubs
Example 2 Semantic features

We might have an ontology giving properties of various nouns/verbs
⇒ how do we allow the parser to use this information?

  pour the **cappucino**
  vs. pour the **book**

Ontology states that **cappucino** has the +liquid feature, **book** does not.

---

**Three Components of Global Linear Models**

- **Φ** is a function that maps a structure \((x, y)\) to a **feature vector** \(\Phi(x, y) \in \mathbb{R}^d\)
- **GEN** is a function that maps an input \(x\) to a set of **candidates** \(\text{GEN}(x)\)
- **W** is a parameter vector (also a member of \(\mathbb{R}^d\))
- Training data is used to set the value of **W**

---

**Component 1: Φ**

- **Φ** maps a candidate to a **feature vector** \(\in \mathbb{R}^d\)
- **Φ** defines the **representation** of a candidate

\[
\downarrow \Phi \quad \langle 1, 0, 2, 0, 15, 5 \rangle
\]

---

**Features**

- A “feature” is a function on a structure, e.g.,
  \(h(x, y) = \text{Number of times} \ A \ \text{is seen in} \ (x, y)\)

\[
\begin{align*}
  h(x_1, y_1) &= 1 \\
  h(x_2, y_2) &= 2
\end{align*}
\]
Another Example

- A “feature” is a function on a structure, e.g.,
  \[ h(x, y) = \begin{cases} 
  1 & \text{if } (x, y) \text{ has an instance of non-parallel coordination} \\
  0 & \text{otherwise} 
\end{cases} \]

\[ h(x_1, y_1) = 0 \]

\[ h(x_2, y_2) = 1 \]

A Third Example

- A “feature” is a function on a structure, e.g.,
  \[ h_1(x, y) = \text{number of times Mary is subject of likes} \]
  \[ h_2(x, y) = \text{number of times Mary is object of likes} \]

\[ h_1(x_1, y_1) = 1 \quad h_2(x_1, y_1) = 1 \]

A Final Example

- A “feature” is a function on a structure, e.g.,
  \[ h(x, y) = \log \text{probability of } (x, y) \text{ under Model 2} \]

\[ h(x_1, y_1) = -1.56 \quad h(x_2, y_2) = -1.98 \]
**Component 1: $\Phi$**

- $\Phi$ maps a candidate to a **feature vector** $\in \mathbb{R}^d$
- $\Phi$ defines the **representation** of a candidate

```
\downarrow \Phi

\langle 1, 0, 2, 0, 15, 5 \rangle
```

---

**Feature Vectors**

- Our goal is to come up with learning methods which allow us to define any features over parse trees
- Avoids the intermediate steps of a history-based model: defining a **derivation** which takes the features into account, and then attaching probabilities to **decisions**
  - Our claim is that this can be an unwieldy, indirect way of getting desired features into a model
- Problem of **representation** now boils down to the choice of the function $\Phi(x, y)$

---

**Feature Vectors**

- A set of functions $h_1 \ldots h_d$ define a **feature vector**
  - $\Phi(x) = \langle h_1(x), h_2(x) \ldots h_d(x) \rangle$

---

**Component 2: GEN**

- GEN enumerates a set of **candidates** for a sentence

```
\downarrow GEN

She announced a program to promote safety in trucks and vans
```

```
\Phi(T_1) = \langle 1, 0, 0, 3 \rangle \quad \Phi(T_2) = \langle 2, 0, 1, 1 \rangle
```
Component 2: GEN

- **GEN** enumerates a set of candidates for an input \( x \)

- Some examples of how **GEN**(\( x \)) can be defined:
  - Parsing: **GEN**(\( x \)) is the set of parses for \( x \) under a grammar
  - Any task: **GEN**(\( x \)) is the top \( N \) most probable parses under a history-based model
  - Tagging: **GEN**(\( x \)) is the set of all possible tag sequences with the same length as \( x \)
  - Translation: **GEN**(\( x \)) is the set of all possible English translations for the French sentence \( x \)

Putting it all Together

- \( \mathcal{X} \) is set of sentences, \( \mathcal{Y} \) is set of possible outputs (e.g. trees)

- Need to learn a function \( F: \mathcal{X} \rightarrow \mathcal{Y} \)

- **GEN**, \( \Phi \), \( W \) define

\[
F(x) = \arg \max_{y \in \text{GEN}(x)} \Phi(x, y) \cdot W
\]

Choose the highest scoring candidate as the most plausible structure

- Given examples \((x_i, y_i)\), how to set \( W \)?

Component 3: \( W \)

- \( W \) is a parameter vector \( \in \mathbb{R}^d \)

- \( \Phi \) and \( W \) together map a candidate to a real-valued score

She announced a program to promote safety in trucks and vans

\[
\downarrow \Phi
\]

\[
(1, 0, 2, 0, 0, 15, 5)
\]

\[
\downarrow \Phi \cdot W
\]

\[
(1, 0, 2, 0, 0, 15, 5) \cdot (1.9, -0.3, 0.2, 1.3, 0, 1.0, -2.3) = 5.8
\]
Overview

- A brief review of history-based methods
- A new framework: Global linear models
- Parsing problems in this framework:
  Reranking problems
- Parameter estimation method 1:
  A variant of the perceptron algorithm

The Representation $\Phi$

- Each component of $\Phi$ could be essentially any feature over parse trees
- For example:
  $\Phi_1(x, y) = \log$ probability of $(x, y)$ under the baseline model
  $\Phi_2(x, y) = \begin{cases} 
  1 & \text{if } (x, y) \text{ includes the rule } VP \rightarrow PP \ VBD \ NP \\
  0 & \text{otherwise}
\end{cases}$

Reranking Approaches to Parsing

- Use a baseline parser to produce top $N$ parses for each sentence in training and test data
  $\text{GEN}(x)$ is the top $N$ parses for $x$ under the baseline model
- One method: use Model 2 to generate a number of parses
  (in our experiments, around 25 parses on average for 40,000 training sentences, giving \( \approx 1 \) million training parses)
- Supervision: for each $x_i$ take $y_i$ to be the parse that is
  “closest” to the treebank parse in $\text{GEN}(x_i)$

Practical Issues in Creating $\Phi$

- Step 1: map a tree to a number of “strings” representing features
  $$\Rightarrow 
  \begin{array}{c}
  \text{HASRULE}: A \rightarrow B; C \\
  \text{HASRULE}: B \rightarrow D; E \\
  \text{HASRULE}: C \rightarrow F; G \\
  \text{HASRULE}: D \rightarrow d \\
  \text{HASRULE}: E \rightarrow e \\
  \text{HASRULE}: F \rightarrow f \\
  \text{HASRULE}: G \rightarrow g
  \end{array}$$

\[70\]
Practical Issues in Creating $\Phi$

- Step 2: hash the strings to integers

```
A  B  C
⇒ HASRULE:A->B;C  54
B  D  E  F  G
⇒ HASRULE:B->D;E  118
⇒ HASRULE:C->F;G  14
⇒ HASRULE:D->d  10078
⇒ HASRULE:E->e  9000
⇒ HASRULE:F->f  1078
⇒ HASRULE:G->g  101
```

- In our experiments, tree is then represented as log probability under the baseline model, plus a sparse feature array:

$$\Phi_i(x, y) = \log \text{probability of } (x, y) \text{ under the baseline model}$$

$$\Phi_i(x, y) = 1 \text{ for } i = \{54, 118, 14, 10078, 9000, 1078, 101\}$$

$$\Phi_i(x, y) = 0 \text{ for all other } i$$

From [Collins and Koo, 2005]:

The following types of features were included in the model. We will use the rule $VP \rightarrow PP \ VBD \ NP \ NP \ SBAR$ with head $VBD$ as an example. Note that the output of our baseline parser produces syntactic trees with headword annotations.

The Score for Each Parse

- In our experiments, tree is then represented as log probability under the baseline model, plus a sparse feature array:

$$\Phi_1(x, y) = \log \text{probability of } (x, y) \text{ under the baseline model}$$

$$\Phi_i(x, y) = 1 \text{ for } i = \{54, 118, 14, 10078, 9000, 1078, 101\}$$

$$\Phi_i(x, y) = 0 \text{ for all other } i$$

- Score for the tree $(x, y)$:

$$\Phi(x, y) \cdot W = \sum_i \Phi_i(x, y)W_i$$

$$= W_1 \Phi_1(x, y) + W_{54} + W_{118} + W_{14} + W_{10078} + W_{9000} + W_{1078} + W_{101}$$

Rules These include all context-free rules in the tree, for example $VP \rightarrow PP \ VBD \ NP \ NP \ SBAR$. 
**Bigrams** These are adjacent pairs of non-terminals to the left and right of the head. As shown, the example rule would contribute the bigrams (Right, VP, NP, NP), (Right, VP, NP, SBAR), (Right, VP, SBAR, STOP), and (Left, VP, PP, STOP) to the left of the head.

**Lexical Bigrams** Same as Bigrams, but with the lexical heads of the two non-terminals also included.

**Grandparent Rules** Same as Rules, but also including the non-terminal above the rule.

**Grandparent Bigrams** Same as Bigrams, but also including the non-terminal above the bigrams.
Two-level Rules Same as Rules, but also including the entire rule above the rule.

Trigrams All trigrams within the rule. The example rule would contribute the trigrams \((VP, STOP, PP, VBD!), (VP, PP, VBD!, NP), (VP, VBD!, NP, NP), (VP, NP, NP, SBAR)\) and \((VP, NP, SBAR, STOP)\) (! is used to mark the head of the rule)

Two-level Bigrams Same as Bigrams, but also including the entire rule above the rule.

Head-Modifiers All head-modifier pairs, with the grandparent non-terminal also included. An \texttt{adj} flag is also included, which is 1 if the modifier is adjacent to the head, 0 otherwise. As an example, say the non-terminal dominating the example rule is S. The example rule would contribute \((\text{Left}, S, VP, VBD, PP, \text{adj}=1)\), \((\text{Right}, S, VP, VBD, NP, \text{adj}=1)\), \((\text{Right}, S, VP, VBD, NP, \text{adj}=0)\), and \((\text{Right}, S, VP, VBD, SBAR, \text{adj}=0)\).
**PPs** Lexical trigrams involving the heads of arguments of prepositional phrases. The example shown at right would contribute the trigram (NP, NP, PP, NP, president, of, U.S.), in addition to the more general trigram relation (NP, NP, PP, NP, of, U.S.).

**Distance Head-Modifiers** Features involving the distance between head words. For example, assume \( \text{dist} \) is the number of words between the head words of the VBD and SBAR in the (VP, VBD, SBAR) head-modifier relation in the above rule. This relation would then generate features (VP, VBD, SBAR, = \( \text{dist} \)), and (VP, VBD, SBAR, \( \leq x \)) for all \( \text{dist} \leq x \leq 9 \) and (VP, VBD, SBAR, \( \geq x \)) for all \( 1 \leq x \leq \text{dist} \).

**Further Lexicalization** In order to generate more features, a second pass was made where all non-terminals were augmented with their lexical heads when these headwords were closed-class words. All features apart from **Head-Modifiers**, **PPs** and **Distance Head-Modifiers** were then generated with these augmented non-terminals.
Overview

- A brief review of history-based methods
- A new framework: Global linear models
- Parsing problems in this framework:
  Reranking problems
- Parameter estimation method 1:
  A variant of the perceptron algorithm

A Variant of the Perceptron Algorithm

Inputs: Training set \((x_i, y_i)\) for \(i = 1 \ldots n\)

Initialization: \(W = 0\)

Define: 
\[ F(x) = \arg \max_{y \in \text{GEN}(x)} \Phi(x, y) \cdot W \]

Algorithm: 
For \(t = 1 \ldots T, i = 1 \ldots n\)
\(z_i = F(x_i)\)
\(\text{If } (z_i \neq y_i) \quad W = W + \Phi(x_i, y_i) - \Phi(x_i, z_i)\)

Output: Parameters \(W\)

Theory Underlying the Algorithm

- Definition: \(\text{GEN}(x_i) = \text{GEN}(x_i) - \{y_i\}\)

- Definition: The training set is separable with margin \(\delta\), if there is a vector \(U \in \mathbb{R}^d\) with \(||U|| = 1\) such that
\[ \forall i, \forall z \in \text{GEN}(x_i) \quad U \cdot \Phi(x_i, y_i) - U \cdot \Phi(x_i, z) \geq \delta \]
**THEORY UNDERLYING THE ALGORITHM**

**Theorem:** For any training sequence \((x_i, y_i)\) which is separable with margin \(\delta\), then for the perceptron algorithm

\[
\text{Number of mistakes} \leq \frac{R^2}{\delta^2}
\]

where \(R\) is a constant such that \(\forall i, \forall z \in \text{GEN}(x_i)\)

\[
||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R
\]

**Proof:** Direct modification of the proof for the classification case.
Proof:
Let $W^k$ be the weights before the $k$'th mistake. $W^1 = 0$
If the $k$'th mistake is made at $i$'th example,
and $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(y) \cdot W^k$, then
$$W^{k+1} = W^k + \Phi(y_i) - \Phi(z_i)$$
$$\Rightarrow U \cdot W^{k+1} = U \cdot W^k + U \cdot \Phi(y_i) - U \cdot \Phi(z_i)$$
$$\geq U \cdot W^k + \delta$$
$$\geq k \delta$$
$$\Rightarrow ||W^{k+1}|| \geq k \delta$$

Also,
$$||W^{k+1}||^2 = ||W^k||^2 + ||\Phi(y_i) - \Phi(z_i)||^2 + 2W^k \cdot (\Phi(y_i) - \Phi(z_i))$$
$$\leq ||W^k||^2 + R^2$$
$$\Rightarrow ||W^{k+1}||^2 \leq kR^2$$
$$\Rightarrow k \delta^2 \leq ||W^{k+1}||^2 \leq kR^2$$
$$\Rightarrow k \leq \frac{R^2}{\delta^2}$$

Summary
- A new framework: global linear models $\text{GEN}, \Phi, W$
- There are several ways to train the parameters $W$:
  - Perceptron
  - Boosting
  - Log-linear models (maximum-likelihood)
- Applications:
  - Reranking models
  - LFG parsing
  - Generation
  - Machine translation
  - Tagging problems
  - Speech recognition

Perceptron Experiments: Parse Reranking

Parsing the Wall Street Journal Treebank
Training set = 40,000 sentences, test = 2,416 sentences
Generative model (Collins 1999): 88.2% F-measure
Reranked model: 89.5% F-measure (11% relative error reduction)
Boosting: 89.7% F-measure (13% relative error reduction)

- Results from Charniak and Johnson, 2005:
  - Improvement from 89.7% (baseline generative model) to 91.0% accuracy
  - Uses a log-linear model
  - Gains from improved n-best lists, better features