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
  - The EM Algorithm
  - 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
    \[
    F(x) = \arg\max_y P(y \mid x, \Theta)
    \]
    Can be computed using dynamic programming

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 $P(\alpha \rightarrow \beta \mid \alpha)$
- Probability of a structure is a product of decision probabilities
  \[
  P(T, S) = \prod_{i=1}^{n} P(\alpha_i \rightarrow \beta_i \mid \alpha_i)
  \]
  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
  \[
  P(\alpha \rightarrow \beta \mid \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}
  \]

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
  \[
  P(t_i \mid t_{i-1}, t_{i-2}, w_1 \ldots w_n)
  \]
  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
  \[
  P(t_1 \ldots t_n \mid w_1 \ldots w_n) = \prod_{i=1}^{n} P(t_i \mid t_{i-1}, t_{i-2}, w_1 \ldots w_n)
  \]
- Parameter values are estimated using variants of maximum-likelihood estimation
  \[
  P(t_i \mid t_{i-1}, t_{i-2}, w_1 \ldots w_n)
  \]
  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

---

**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**

---

**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

---

**An Example: Parsing**

- In lecture 4, we described lexicalized models for parsing

- Showed how a rule can be generated in a number of steps

![Diagram](image-url)
Model 2

• Step 1: generate category of head child

\[ S(\text{told}, V[6]) \]

\[ \downarrow \]

\[ S(\text{told}, V[6]) \]

\[ \downarrow \]

\[ \text{VP}(\text{told}, V[6]) \]

\[ P_h(\text{VP} | \text{S, told, V}[6]) \]

13

Model 2

• Step 2: choose left subcategorization frame

\[ S(\text{told}, V[6]) \]

\[ \downarrow \]

\[ S(\text{told}, V[6]) \]

\[ \downarrow \]

\[ \text{VP}(\text{told}, V[6]) \]

\[ \{\text{NP-C}\} \]

\[ P_h(\text{VP} | \text{S, told, V}[6]) \times P_e(\{\text{NP-C}\} | \text{S, VP, told, V}[6]) \]

\[ P_d(\text{NP-C}(\text{Hillary}, \text{NNP}) | \text{S, VP, told, V}[6], \text{LEFT}, \{\text{NP-C}\}) \]

14

• Step 3: generate left modifiers in a Markov chain

\[ S(\text{told}, V[6]) \]

\[ \downarrow \]

\[ \text{NP-C}(\text{Hillary}, \text{NNP}) \]

\[ \text{VP}(\text{told}, V[6]) \]

\[ \{\} \]

\[ P_h(\text{VP} | \text{S, told, V}[6]) \times P_e(\{\text{NP-C}\} | \text{S, VP, told, V}[6]) \]

\[ P_d(\text{NP-C}(\text{Hillary}, \text{NNP}) | \text{S, VP, told, V}[6], \text{LEFT}, \{\text{NP-C}\}) \]

\[ P_d(\text{NP}(\text{yesterday}, \text{NN}) | \text{S, VP, told, V}[6], \text{LEFT}, \{\}) \]

15
### The Probabilities for One Rule

\[
P(S(told,V[6])) =
\]
\[
P_h(VP | S, told, V[6]) \times P_c(({NP-C} | S, VP, told, V[6]) \times

P_d(NP(yesterday,NN) | S, VP, told, V[6], LEFT, {})) \times

P_d(NP(C(Hillary,NNP) | S, VP, told, V[6], LEFT, {})) \times

P_d(S(told,V[6]) | S, VP, told, V[6], LEFT, {})
\]

**Three parameter types:**
- Head parameters
- Subcategorization parameters
- Dependency parameters

### Smoothed Estimation

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

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

---

\[
P(lawyer | S, VP, NP, NN, questioned, Vt) =
\]
\[
\lambda_1 \times \frac{\text{Count}(lawyer | S, VP, NP, NN, questioned, Vt)}{\text{Count}(S, VP, NP, NN, questioned, Vt)}
\]
\[
+ \lambda_2 \times \frac{\text{Count}(lawyer | S, VP, NP, NN, Vt)}{\text{Count}(S, VP, NP, NN, Vt)}
\]
\[
+ \lambda_3 \times \frac{\text{Count}(lawyer | NN)}{\text{Count}(NN)}
\]

- Where \(0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 1\), and \(\lambda_1 + \lambda_2 + \lambda_3 = 1\)
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

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.

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

Three Components of Global Linear Models

- \( f \) is a function that maps a structure \((x, y)\) to a feature vector \( f(x, y) \in \mathbb{R}^d \)
- \( \text{GEN} \) 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: \( f \)

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

\[
\begin{align*}
\downarrow f \\
\langle 1, 0, 2, 0, 0, 15, 5 \rangle
\end{align*}
\]

---

Another Example

- A “feature” is a function on a structure, e.g.,

\[
h(x, y) = \begin{cases} 
1 & \text{if } (x, y) \text{ has an instance of non-parallel coordination} \\
0 & \text{otherwise} 
\end{cases}
\]

\[
(x_1, y_1) \\
\text{visited} \\
\text{NP} \\
\text{and} \\
\text{NP} \\
\text{bars in New York} \\
\text{pubs in London}
\]

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

---

Features

- A “feature” is a function on a structure, e.g.,

\[
h(x, y) = \text{Number of times } \begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array} \text{ is seen in } (x, y)
\]

\[
(x_1, y_1) \\
\text{A} \\
\text{B} \\
\text{C} \\
\text{D E F G}
\]

\[
(x_2, y_2) \\
\text{A} \\
\text{B} \\
\text{C} \\
\text{D E F A}
\]

\[
h(x_1, y_1) = 1 \\
h(x_2, y_2) = 2
\]
**A Third Example**

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

\[
\begin{array}{c}
(x, y) \\
\hspace{1cm} S \\
\hspace{2cm} NP \\
\hspace{3cm} Jim \\
\hspace{4cm} likes \\
\hspace{5cm} NP \\
\hspace{6cm} SBAR \\
\hspace{7cm} who \\
\hspace{8cm} likes \\
\hspace{9cm} Bill \\
\end{array}
\]

\[ h_1(x, y) = 1 \quad h_2(x, y) = 1 \]

---

**Component 1: f**

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

\[
\rightarrow f
\]

\[ \langle 1, 0, 2, 0, 15, 5 \rangle \]

---

**A Final Example**

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

\[
\begin{array}{c}
(x_1, y_1) \\
\hspace{1cm} A \\
\hspace{2cm} B \\
\hspace{3cm} C \\
\hspace{4cm} d \\
\hspace{5cm} e \\
\hspace{6cm} f \\
\hspace{7cm} g \\
\end{array}
\]

\[ h(x_1, y_1) = -1.56 \]

\[
\begin{array}{c}
(x_2, y_2) \\
\hspace{1cm} A \\
\hspace{2cm} B \\
\hspace{3cm} C \\
\hspace{4cm} d \\
\hspace{5cm} e \\
\hspace{6cm} h \\
\hspace{7cm} B \\
\hspace{8cm} C \\
\hspace{9cm} b \\
\hspace{10cm} c \\
\end{array}
\]

\[ h(x_2, y_2) = -1.98 \]

---

**Feature Vectors**

- A set of functions \( h_1 \ldots h_d \) define a feature vector
  
  \[ f(x) = \langle h_1(x), h_2(x) \ldots h_d(x) \rangle \]

\[
\begin{array}{c}
T_1 \\
\hspace{1cm} A \\
\hspace{2cm} B \\
\hspace{3cm} C \\
\hspace{4cm} D \\
\hspace{5cm} E \\
\hspace{6cm} F \\
\hspace{7cm} G \\
\end{array}
\]

\[ f(T_1) = \langle 1, 0, 0, 3 \rangle \]

\[
\begin{array}{c}
T_2 \\
\hspace{1cm} A \\
\hspace{2cm} B \\
\hspace{3cm} C \\
\hspace{4cm} D \\
\hspace{5cm} E \\
\hspace{6cm} F \\
\hspace{7cm} G \\
\end{array}
\]

\[ f(T_2) = \langle 2, 0, 1, 1 \rangle \]
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 $f(x, y)$.

Component 2: GEN

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

Component 3: W

- $W$ is a parameter vector $\in \mathbb{R}^d$.
- $f$ and $W$ together map a candidate to a real-valued score.

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

\[ \downarrow \text{GEN} \]

\[ \begin{align*}
  &\downarrow f \\
  &\langle 1, 0, 2, 0, 0, 15, 5 \rangle \\
  &\downarrow f \cdot W \\
  &\langle 1, 0, 2, 0, 0, 15, 5 \rangle \cdot \langle 1.9, -0.3, 0.2, 1.3, 0, 1.0, -2.3 \rangle = 5.8
\]
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 $F : \mathcal{X} \rightarrow \mathcal{Y}$
- $\text{GEN}$, $f$, $W$ define
  \[ F(x) = \arg \max_{y \in \text{GEN}(x)} f(x, y) \cdot W \]

Choose the highest scoring candidate as the most plausible structure

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

She announced a program to promote safety in trucks and vans

\[ \downarrow \text{GEN} \]

\[ \downarrow f \quad \downarrow f \quad \downarrow f \quad \downarrow f \quad \downarrow f \quad \downarrow f \]

(1, 1, 3, 5) (2, 0, 0, 5) (1, 0, 1, 5) (0, 0, 3, 0) (0, 1, 0, 5) (0, 0, 1, 5)

\[ \downarrow f \cdot W \quad \downarrow f \cdot W \quad \downarrow f \cdot W \quad \downarrow f \cdot W \quad \downarrow f \cdot W \quad \downarrow f \cdot W \]

13.6 12.2 12.1 3.3 9.4 11.1

\[ \downarrow \arg \max \]

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

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 $\text{GEN}(x_i)$
The Representation $f$

- Each component of $f$ could be essentially any feature over parse trees.

- For example:
  
  \[
  f_1(x, y) = \log \text{probability of } (x, y) \text{ under the baseline model}
  \]

  \[
  f_2(x, y) = \begin{cases} 
  1 & \text{if } (x, y) \text{ includes the rule } VP \rightarrow PP \ VBD \ NP \\
  0 & \text{otherwise}
  \end{cases}
  \]

Practical Issues in Creating $f$

- Step 1: map a tree to a number of “strings” representing features

  \[
  \begin{array}{c}
  A \\
  \downarrow \\
  B \\
  \downarrow \\
  C \\
  \downarrow \\
  D \quad E \quad F \quad G \\
  \downarrow \downarrow \downarrow \\
  d \quad e \quad f \quad g
  \end{array}
  \Rightarrow
  \begin{array}{cc}
  \text{HASRULE:} & A \rightarrow B; C \\
  \text{HASRULE:} & B \rightarrow D; E \\
  \text{HASRULE:} & C \rightarrow F; G \\
  \text{HASRULE:} & D \rightarrow d \\
  \text{HASRULE:} & E \rightarrow e \\
  \text{HASRULE:} & F \rightarrow f \\
  \text{HASRULE:} & G \rightarrow g
  \end{array}
  \]

- Step 2: hash the strings to integers

  \[
  A \rightarrow B; C \quad 54
  \]
  \[
  B \rightarrow D; E \quad 118
  \]
  \[
  C \rightarrow F; G \quad 14
  \]
  \[
  D \rightarrow d \quad 10078
  \]
  \[
  E \rightarrow e \quad 9000
  \]
  \[
  F \rightarrow f \quad 1078
  \]
  \[
  G \rightarrow g \quad 101
  \]

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

\[
\begin{align*}
  f_1(x, y) &= \log \text{probability of } (x, y) \text{ under the baseline model} \\
  f_i(x, y) &= 1 \text{ for } i = \{54, 118, 14, 10078, 9000, 1078, 101\} \\
  f_i(x, y) &= 0 \text{ for all other } i
\end{align*}
\]

The Score for Each Parse

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

  \[
  f_1(x, y) = \log \text{probability of } (x, y) \text{ under the baseline model} \\
  f_i(x, y) = 1 \text{ for } i = \{54, 118, 14, 10078, 9000, 1078, 101\} \\
  f_i(x, y) = 0 \text{ for all other } i
  \]

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

\[
\begin{align*}
  f(x, y) \cdot W \\
  &= \sum_{i} f_i(x, y) W_i \\
  &= W_1 f_1(x, y) + W_{54} + W_{118} + W_{14} + W_{10078} + W_{9000} + W_{1078} + W_{101}
\end{align*}
\]
From [Collins and Koo, 2005]:
The following types of features were included in the model. We will use the rule VP → 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.

**Rules** These include all context-free rules in the tree, for example VP → 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.

**Grandparent Rules** Same as Rules, but also including the non-terminal above the rule.
Lexical Bigrams Same as Bigrams, but with the lexical heads of the two non-terminals also included.

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.

Two-level Bigrams Same as Bigrams, 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)

![Trigram Diagram](image)

**Head-Modifiers** All head-modifier pairs, with the grandparent non-terminal also included. An 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 \((Left, S, VP, VBD, PP, adj=1)\), \((Right, S, VP, VBD, NP, adj=1)\), \((Right, S, VP, VBD, NP, adj=0)\), and \((Right, S, VP, VBD, SBAR, adj=0)\).

![Head-Modifier Diagram](image)

**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 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, = dist), and (VP, VBD, SBAR, ≤ x) for all dist ≤ x ≤ 9 and (VP, VBD, SBAR, ≥ x) for all 1 ≤ x ≤ 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: \[ F(x) = \operatorname{argmax}_{y \in \text{GEN}(x)} f(x, y) \cdot W \]

Algorithm:

For \(t = 1 \ldots T, i = 1 \ldots n\)

\[ z_i = F(x_i) \]

If \(z_i \neq y_i\) \(W = W + f(x_i, y_i) - f(x_i, z_i)\)

Output: Parameters \(W\)
Say first tree is correct, but second tree has higher $W \cdot f(x, y)$ under current parameters: Then $W = W + f(x_i, y_i) - f(x_i, z_i)$ implies

$$
W_1 \quad + = \quad -1.56 - (-1.13) = -0.43 \\
W_{54} \quad + = \quad 1 \\
W_{89} \quad + = \quad -1
$$

**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

\[ \forall i, \forall z \in \text{GEN}(x_i) \quad U \cdot f(x_i, y_i) - U \cdot f(x_i, z) \geq \delta \]
Direct modification of the proof for the classification case.

Uses a log-linear model

91.0% accuracy

Improvement from 89.7% (baseline generative model) to

\[ \text{Theorem:} \text{ For any training sequence } (x_i, y_i) \text{ which is separable with margin } \delta, \text{ then for the perceptron algorithm} \]

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

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

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

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

\[ z_i = \arg \max_{y \in \text{GEN}(x_i)} f(x_i, y) \cdot W^k, \text{ then} \]

\[ W^{k+1} = W^k + f(x_i, y_i) - f(x_i, z_i) \]

\[ \Rightarrow U \cdot W^{k+1} = U \cdot W^k + U \cdot f(x_i, y_i) - U \cdot f(x_i, z_i) \]

\[ \geq U \cdot W^k + \delta \]

\[ \Rightarrow ||W^{k+1}|| \geq k\delta \]

Also,

\[ ||W^{k+1}||^2 = ||W^k||^2 + ||f(x_i, y_i) - f(x_i, z_i)||^2 + 2W^k \cdot (f(x_i, y_i) - f(x_i, z_i)) \]

\[ \leq ||W^k||^2 + R^2 \]

\[ \Rightarrow ||W^{k+1}||^2 \leq kR^2 \]

\[ \Rightarrow k^2\delta^2 \leq ||W^{k+1}||^2 \leq kR^2 \]

\[ \Rightarrow k \leq \frac{R^2}{\delta^2} \]

Theory Underlying the Algorithm

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
Summary

- A new framework: **global linear models**
  \[\text{GEN}, f, W\]

- 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