Discriminative Training I: Intro & PRO

April 3, 2014
Noisy Channels Again

$p(e)$

source $\rightarrow$ English
Noisy Channels Again

$p(e) \xrightarrow{} \text{English} \xrightarrow{} p(g \mid e) \xrightarrow{} \text{German}$
Noisy Channels Again

\[ p(e) \]

source → English

\[ p(g | e) \]

decoder

German

\[ e^* = \arg \max_e p(e | g) \]

\[ = \arg \max_e \frac{p(g | e) \times p(e)}{p(g)} \]

\[ = \arg \max_e p(g | e) \times p(e) \]
Noisy Channels Again

\[
\begin{align*}
e^* &= \arg \max_e p(e \mid g) \\
     &= \arg \max_e \frac{p(g \mid e) \times p(e)}{p(g)} \\
     &= \arg \max_e p(g \mid e) \times p(e)
\end{align*}
\]
Noisy Channels Again

\[ e^* = \arg \max_e p(e \mid g) \]
\[ = \arg \max_e \frac{p(g \mid e) \times p(e)}{p(g)} \]
\[ = \arg \max_e p(g \mid e) \times p(e) \]
\[ = \arg \max_e \log p(g \mid e) + \log p(e) \]
Noisy Channels Again

\[ e^* = \arg \max_e p(e \mid g) \]

\[ = \arg \max_e \frac{p(g \mid e) \times p(e)}{p(g)} \]

\[ = \arg \max_e p(g \mid e) \times p(e) \]

\[ = \arg \max_e \log p(g \mid e) + \log p(e) \]

Does this look familiar?

\[ = \arg \max_e \begin{bmatrix} 1 \\ 1 \end{bmatrix}^\top \begin{bmatrix} \log p(g \mid e) \\ \log p(e) \end{bmatrix} \]

\[ w^\top \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]

\[ h(g,e) \]
The Noisy Channel

\[-\log p(g | e)\]

\[-\log p(e)\]
As a Linear Model

\[-\log p(g \mid e)\]
As a Linear Model

\[-\log p(g|e)\] vs. \[-\log p(e)\]

\[\vec{w}\]
As a Linear Model

\[-\log p(g|e)\]

\[-\log p(e)\]
As a Linear Model

\[-\log p(g | e)\]

Improvement 1:
change \(\vec{w}\) to find better translations
As a Linear Model

\[-\log p(g|e) \sim \vec{w}\]
As a Linear Model

\[-\log p(g|e)\]

\[-\log p(e)\]
As a Linear Model

\[-\log p(g|e)\]

\[-\log p(e)\]
As a Linear Model

-\log p(g \mid e)

Improvement 2:
Add dimensions to make points separable
Linear Models

\[ e^* = \arg \max_e w^\top h(g, e) \]

- Improve the modeling capacity of the noisy channel in two ways
  - Reorient the weight vector
  - Add new dimensions (new features)
- Questions
  - What features? \( h(g, e) \)
  - How do we set the weights? \( w \)
Mann beißt Hund

x BITES y
Mann beißt Hund

man bites cat

Mann beißt Hund

man chase dog

Mann beißt Hund

dog man bites

Mann beißt Hund

man bite dog

Mann beißt Hund

dog man bites

Mann beißt Hund

man bites dog

Mann beißt Hund

man chase dog

Mann beißt Hund

dog man bites
Feature Classes

Lexical
Are lexical choices appropriate?
\(\text{bank} = \text{“River bank” vs. “Financial institution”}\)

Configurational
Are semantic/syntactic relations preserved?
“Dog bites man” vs. “Man bites dog”

Grammatical
Is the output fluent / well-formed?
“Man \textit{bites} dog” vs. “Man \textit{bite} dog”
What do lexical features look like?

First attempt:

\[
\text{score}(\text{g}, \text{e}) = \mathbf{w}^\top \mathbf{h}(\text{g}, \text{e})
\]

\[
h_{15,342}(\text{g}, \text{e}) = \begin{cases} 
1, & \exists i, j : g_i = \text{Hund}, e_j = \text{cat} \\
0, & \text{otherwise}
\end{cases}
\]

But what if a cat is being chased by a Hund?
What do lexical features look like?

Latent variables enable more precise features:

\[
score(g, e, a) = w^\top h(g, e, a)
\]

\[
h_{15,342}(g, e, a) = \sum_{(i,j) \in a} \begin{cases} 
1, & \text{if } g_i = \text{Hund}, e_j = \text{cat} \\
0, & \text{otherwise}
\end{cases}
\]
Standard Features

• Target side features
  • $\log p(e)$  
    [ *n*-gram language model *]
  • Number of words in hypothesis
  • Non-English character count

• Source + target features
  • $\log$ relative frequency $e|f$ of each rule  
    [ $\log \#(e,f) - \log \#(f)$ ]
  • $\log$ relative frequency $f|e$ of each rule  
    [ $\log \#(e,f) - \log \#(e)$ ]
  • “lexical translation” log probability $e|f$ of each rule  
    [ $\approx \log p_{\text{model1}}(e|f)$ ]
  • “lexical translation” log probability $f|e$ of each rule  
    [ $\approx \log p_{\text{model1}}(f|e)$ ]

• Other features
  • Count of rules/phrases used
  • Reordering pattern probabilities
Feature Locality

• Dynamic programming recombination assumes that features are “rule local”

• The must have the same value independent of the other rules that are used around them

• Features that look at “large amounts of structure” are expensive to compute

• Language models are “medium sized” features
Why do this?

Table 2: Effect of maximum entropy training for alignment template approach (WP: word penalty feature, CLM: class-based language model (five-gram), MX: conventional dictionary).

<table>
<thead>
<tr>
<th></th>
<th>objective criteria [%]</th>
<th>subjective criteria [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SER</td>
<td>WER</td>
</tr>
<tr>
<td>Baseline($\lambda_m = 1$)</td>
<td>86.9</td>
<td>42.8</td>
</tr>
<tr>
<td>ME</td>
<td>81.7</td>
<td>40.2</td>
</tr>
<tr>
<td>ME+WP</td>
<td>80.5</td>
<td>38.6</td>
</tr>
<tr>
<td>ME+WP+CLM</td>
<td>78.1</td>
<td>38.3</td>
</tr>
<tr>
<td>ME+WP+CLM+MX</td>
<td>77.8</td>
<td>38.4</td>
</tr>
</tbody>
</table>

Discriminative
Parameter Learning
Hypothesis Space
Hypothesis Space
Preliminaries

We assume a decoder that computes:

\[ \langle e^*, a^* \rangle = \arg \max_{\langle e, a \rangle} w^T h(g, e, a) \]

And \( K \)-best lists of, that is:

\[ \{ \langle e_i^*, a_i^* \rangle \}_{i=1}^{K} = \arg i^{th} \max_{\langle e, a \rangle} w^T h(g, e, a) \]

Standard, efficient algorithms exist for this.
Learning Weights

- Try to match the reference translation exactly
- Conditional random field
  - Maximize the conditional probability of the reference translations
  - “Average” over the different latent variables
Problems

• These methods give “full credit” when the model exactly produces the reference and no credit otherwise

• What is the problem with this?
Cost-Sensitive Training

• Assume we have a cost function that gives a score for how good/bad a translation is

\[ \ell(\hat{e}, E) \mapsto [0, 1] \]

• Optimize the weight vector by making reference to this function

• We will talk about two ways to do this
K-Best List Example
K-Best List Example
Training as Classification

- Pairwise Ranking Optimization
  - Reduce training problem to **binary classification** with a linear model

- Algorithm
  - For $i=1$ to $N$
    - Pick random pair of hypotheses (A,B) from $K$-best list
    - Use cost function to determine if is A or B better
    - Create $i$th training instance
  - Train binary linear classifier
\[ h_1 \]
\[ h_2 \]

- \( 0.8 \leq \ell < 1.0 \)
- \( 0.6 \leq \ell < 0.8 \)
- \( 0.4 \leq \ell < 0.6 \)
- \( 0.2 \leq \ell < 0.4 \)
- \( 0.0 \leq \ell < 0.2 \)
Worse!

- 0.8 ≤ ℓ < 1.0
- 0.6 ≤ ℓ < 0.8
- 0.4 ≤ ℓ < 0.6
- 0.2 ≤ ℓ < 0.4
- 0.0 ≤ ℓ < 0.2
Worse!

- $0.8 \leq \ell < 1.0$
- $0.6 \leq \ell < 0.8$
- $0.4 \leq \ell < 0.6$
- $0.2 \leq \ell < 0.4$
- $0.0 \leq \ell < 0.2$
Better!

- 0.8 \leq \ell < 1.0
- 0.6 \leq \ell < 0.8
- 0.4 \leq \ell < 0.6
- 0.2 \leq \ell < 0.4
- 0.0 \leq \ell < 0.2
Better!
Worse!

- 0.8 ≤ ℓ < 1.0
- 0.6 ≤ ℓ < 0.8
- 0.4 ≤ ℓ < 0.6
- 0.2 ≤ ℓ < 0.4
- 0.0 ≤ ℓ < 0.2
Better!
Fit a linear model
Fit a linear model
K-Best List Example

\[ w \]

\[ h_1 \]

\[ h_2 \]

- \( 0.8 \leq \ell < 1.0 \)
- \( 0.6 \leq \ell < 0.8 \)
- \( 0.4 \leq \ell < 0.6 \)
- \( 0.2 \leq \ell < 0.4 \)
- \( 0.0 \leq \ell < 0.2 \)