Data Intensive Linguistics — Lecture 15
Machine translation (II): Word-based models and the EM algorithm

Philipp Koehn

2 March 2006
Lexical translation

• How to translate a word → look up in dictionary

  **Haus** — *house, building, home, household, shell.*

• *Multiple translations*
  
  – some more frequent than others
  
  – for instance: *house*, and *building* most common
  
  – special cases: *Haus* of a *snail* is its *shell*

• Note: During all the lectures, we will translate from a foreign language into English
Collect statistics

- Look at a parallel corpus (German text along with English translation)

<table>
<thead>
<tr>
<th>Translation of Haus</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>house</td>
<td>8,000</td>
</tr>
<tr>
<td>building</td>
<td>1,600</td>
</tr>
<tr>
<td>home</td>
<td>200</td>
</tr>
<tr>
<td>household</td>
<td>150</td>
</tr>
<tr>
<td>shell</td>
<td>50</td>
</tr>
</tbody>
</table>
Estimate translation probabilities

- **Maximum likelihood estimation**

$$p_f(e) = \begin{cases} 
0.8 & \text{if } e = \text{house}, \\
0.16 & \text{if } e = \text{building}, \\
0.02 & \text{if } e = \text{home}, \\
0.015 & \text{if } e = \text{household}, \\
0.005 & \text{if } e = \text{shell}.
\end{cases}$$
Alignment

- In a parallel text (or when we translate), we align words in one language with the words in the other

```
1 2 3 4
das Haus ist klein
```
```
the house is small
```

- Word positions are numbered 1–4
Alignment function

• Formalizing *alignment* with an **alignment function**

• Mapping an English target word at position $i$ to a German source word at position $j$ with a function $a : i \rightarrow j$

• Example

\[
a : \{1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, 4 \rightarrow 4\}
\]
Reordering

- Words may be reordered during translation

\[ a : \{1 \rightarrow 3, 2 \rightarrow 4, 3 \rightarrow 2, 4 \rightarrow 1\} \]
One-to-many translation

• A source word may translate into multiple target words

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{das} & \text{Haus} & \text{ist} & \text{klitzeklein} \\
& \text{the} & \text{house} & \text{is} & \text{very} & \text{small} \\
1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[
a : \{1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4, 4 \to 5\}
\]
Dropping words

- Words may be **dropped** when translated
  - The German article *das* is dropped

```
das Haus ist klein
```

```
house is small
```

\[
a : \{1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4\}
\]
Inserting words

• Words may be **added** during translation
  
  – The English *just* does not have an equivalent in German
  – We still need to map it to something: special **NULL** token

\[
a : \{1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, 4 \rightarrow 0, 5 \rightarrow 4\}
\]
IBM Model 1

- **Generative model**: break up translation process into smaller steps
  - IBM Model 1 only uses *lexical translation*

- Translation probability
  - for a foreign sentence $f = (f_1, ..., f_{l_f})$ of length $l_f$
  - to an English sentence $e = (e_1, ..., e_{l_e})$ of length $l_e$
  - with an alignment of each English word $e_j$ to a foreign word $f_i$ according to the alignment function $a : j \rightarrow i$

  $$p(e, a|f) = \frac{\epsilon}{(l_f + 1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})$$

  - parameter $\epsilon$ is a *normalization constant*
Example

<table>
<thead>
<tr>
<th>das</th>
<th></th>
<th>Haus</th>
<th></th>
<th>ist</th>
<th></th>
<th>klein</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>t(e</td>
<td>f)</td>
<td>e</td>
<td>t(e</td>
<td>f)</td>
<td>e</td>
<td>t(e</td>
</tr>
<tr>
<td>the</td>
<td>0.7</td>
<td>house</td>
<td>0.8</td>
<td>is</td>
<td>0.8</td>
<td>small</td>
<td>0.4</td>
</tr>
<tr>
<td>that</td>
<td>0.15</td>
<td>building</td>
<td>0.16</td>
<td>’s</td>
<td>0.16</td>
<td>little</td>
<td>0.4</td>
</tr>
<tr>
<td>which</td>
<td>0.075</td>
<td>home</td>
<td>0.02</td>
<td>exists</td>
<td>0.02</td>
<td>short</td>
<td>0.1</td>
</tr>
<tr>
<td>who</td>
<td>0.05</td>
<td>household</td>
<td>0.015</td>
<td>has</td>
<td>0.015</td>
<td>minor</td>
<td>0.06</td>
</tr>
<tr>
<td>this</td>
<td>0.025</td>
<td>shell</td>
<td>0.005</td>
<td>are</td>
<td>0.005</td>
<td>petty</td>
<td>0.04</td>
</tr>
</tbody>
</table>

\[
p(e, a|f) = \frac{\epsilon}{4^3} \times t(\text{the}|\text{das}) \times t(\text{house}|\text{Haus}) \times t(\text{is}|\text{ist}) \times t(\text{small}|\text{klein})
\]

\[
= \frac{\epsilon}{4^3} \times 0.7 \times 0.8 \times 0.8 \times 0.4
\]

\[
= 0.0028\epsilon
\]
Learning lexical translation models

• We would like to *estimate* the lexical translation probabilities $t(e|f)$ from a parallel corpus

• ... but we do not have the alignments

• **Chicken and egg problem**
  - if we had the *alignments*,
    → we could estimate the *parameters* of our generative model
  - if we had the *parameters*,
    → we could estimate the *alignments*
EM algorithm

- **Incomplete data**
  - if we had *complete data*, would could estimate *model*
  - if we had *model*, we could fill in the *gaps in the data*

- **Expectation Maximization (EM) in a nutshell**
  - initialize model parameters (e.g. uniform)
  - assign probabilities to the missing data
  - estimate model parameters from completed data
  - iterate
**EM algorithm**

... la maison ... la maison blue ... la fleur ...  

... the house ... the blue house ... the flower ...  

- Initial step: all alignments equally likely

- Model learns that, e.g., *la* is often aligned with *the*
EM algorithm

... la maison ... la maison blue ... la fleur ...

... the house ... the blue house ... the flower ...

• After one iteration

• Alignments, e.g., between *la* and *the* are more likely
EM algorithm

... la maison ... la maison bleu ... la fleur ...

... the house ... the blue house ... the flower ...

- After another iteration

- It becomes apparent that alignments, e.g., between *fleur* and *flower* are more likely (pigeon hole principle)
EM algorithm

... la maison ... la maison bleu ... la fleur ...

... the house ... the blue house ... the flower ...

- Convergence

- Inherent hidden structure revealed by EM
EM algorithm

... la maison ... la maison bleu ... la fleur ...

\[ \begin{array}{c}
\text{... the house ... the blue house ... the flower ...} \\
\text{\[ p(la \mid \text{the}) = 0.453 \]
\[ p(le \mid \text{the}) = 0.334 \]
\[ p(\text{maison} \mid \text{house}) = 0.876 \]
\[ p(\text{bleu} \mid \text{blue}) = 0.563 \]
\]}

- Parameter estimation from the aligned corpus
IBM Model 1 and EM

• EM Algorithm consists of two steps

• **Expectation-Step**: Apply model to the data
  – parts of the model are hidden (here: alignments)
  – using the model, assign probabilities to possible values

• **Maximization-Step**: Estimate model from data
  – take assign values as fact
  – collect counts (weighted by probabilities)
  – estimate model from counts

• Iterate these steps until convergence
IBM Model 1 and EM

- We need to be able to compute:
  - Expectation-Step: probability of alignments
  - Maximization-Step: count collection
IBM Model 1 and EM

- **Probabilities**
  
  \[
  p(\text{the}|\text{la}) = 0.7 \\
  p(\text{house}|\text{la}) = 0.05 \\
  p(\text{the}|\text{maison}) = 0.1 \\
  p(\text{house}|\text{maison}) = 0.8
  \]

- **Alignments**

  la \rightarrow \text{the} \rightarrow \text{maison} \rightarrow \text{house} \\
  la \rightarrow \text{the} \rightarrow \text{maison} \rightarrow \text{house} \\
  la \rightarrow \text{the} \rightarrow \text{maison} \rightarrow \text{house} \\
  la \rightarrow \text{the} \rightarrow \text{maison} \rightarrow \text{house}

  \[
  p(a) = 0.56 \\
  p(a) = 0.035 \\
  p(a) = 0.08 \\
  p(a) = 0.005
  \]

  0.824 + 0.052 + 0.118 + 0.007

- **Counts**

  \[
  c(\text{the}|\text{la}) = 0.824 + 0.052 \\
  c(\text{house}|\text{la}) = 0.052 + 0.007 \\
  c(\text{the}|\text{maison}) = 0.118 + 0.007 \\
  c(\text{house}|\text{maison}) = 0.824 + 0.118
  \]
IBM Model 1 and EM: Expectation Step

- We need to compute $p(a|e, f)$

- Applying the *chain rule*:

\[
p(a|e, f) = \frac{p(e, a|f)}{p(e|f)}
\]

- We already have the formula for $p(e, a|f)$ (definition of Model 1)
IBM Model 1 and EM: Expectation Step

- We need to compute $p(e|f)$

$$p(e|f) = \sum_a p(e, a|f)$$

$$= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} p(e, a|f)$$

$$= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \frac{\epsilon}{(l_f + 1)l_e} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})$$
IBM Model 1 and EM: Expectation Step

\[ p(e|f) = \sum_{a(1)=0}^{l_f} \ldots \sum_{a(l_e)=0}^{l_f} \frac{\epsilon}{(l_f + 1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \]

\[ = \frac{\epsilon}{(l_f + 1)^{l_e}} \sum_{a(1)=0}^{l_f} \ldots \sum_{a(l_e)=0}^{l_f} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \]

\[ = \frac{\epsilon}{(l_f + 1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i) \]

- Note the trick in the last line
  - removes the need for an exponential number of products
  → this makes IBM Model 1 estimation tractable
IBM Model 1 and EM: Expectation Step

• Combine what we have:

\[ p(a|e, f) = \frac{p(e, a|f)}{p(e|f)} \]

\[ = \frac{\epsilon}{(l_f+1)^l_e} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) }{ \epsilon \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i) } \]

\[ = \prod_{j=1}^{l_e} \frac{t(e_j|f_{a(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)} \]
IBM Model 1 and EM: Maximization Step

- Now we have to *collect counts*

- Evidence from a sentence pair $e, f$ that word $e$ is a translation of word $f$:

  \[
  c(e|f; e, f) = \sum_a p(a|e, f) \sum_{j=1}^{l_e} \delta(e, e_j) \delta(f, f_{a(j)})
  \]

- With the same simplification as before:

  \[
  c(e|f; e, f) = \frac{t(e|f)}{\sum_{j=1}^{l_e} t(e|f_{a(j)})} \sum_{j=1}^{l_e} \delta(e, e_j) \sum_{i=0}^{l_f} \delta(f, f_i)
  \]
IBM Model 1 and EM: Maximization Step

- After collecting these counts over a corpus, we can estimate the model:

\[
t(e|f; e, f) = \frac{\sum_{(e,f)} c(e|f; e, f))}{\sum_{f} \sum_{(e,f)} c(e|f; e, f))}
\]
IBM Model 1 and EM: Pseudocode

initialize \( t(e|f) \) uniformly

do
  set count(e|f) to 0 for all e,f
  set total(f) to 0 for all f
  for all sentence pairs \((e_s,f_s)\)
    for all words e in e_s
      total_s = 0
      for all words f in f_s
        total_s += t(e|f)
        for all words e in e_s
          for all words f in f_s
            count(e|f) += t(e|f) / total_s
            total(f) += t(e|f) / total_s
      for all f in domain( total(.) )
      for all e in domain( count(.|f) )
        t(e|f) = count(e|f) / total(f)
  until convergence
Higher IBM Models

<table>
<thead>
<tr>
<th>IBM Model</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>lexical translation</td>
</tr>
<tr>
<td>Model 2</td>
<td>adds absolute reordering model</td>
</tr>
<tr>
<td>Model 3</td>
<td>adds fertility model</td>
</tr>
<tr>
<td>Model 4</td>
<td>relative reordering model</td>
</tr>
<tr>
<td>Model 5</td>
<td>fixes deficiency</td>
</tr>
</tbody>
</table>

- Only IBM Model 1 has *global maximum*
  - training of a higher IBM model builds on previous model

- Computationally biggest change in Model 3
  - trick to simplify estimation does not work anymore
  - *exhaustive* count collection becomes computationally too expensive
  - *sampling* over high probability alignments is used instead
IBM Model 4

Mary did not slap the green witch

Mary not slap slap slap the green witch

Mary not slap slap slap NULL the green witch

Maria no daba una bofetada a la verde bruja

Maria no daba una bofetada a la bruja verde

n(3|slap)
p-null
t(la|the)
d(4|4)