MT Marathon
Statistical machine translation: IBM Models and word alignment

Alexandra Birch and Philipp Koehn

January 2010

School of informatics

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)

Translation of $Haus$	**Count**
house	8,000
building	1,600
home	200
household	150
shell	50

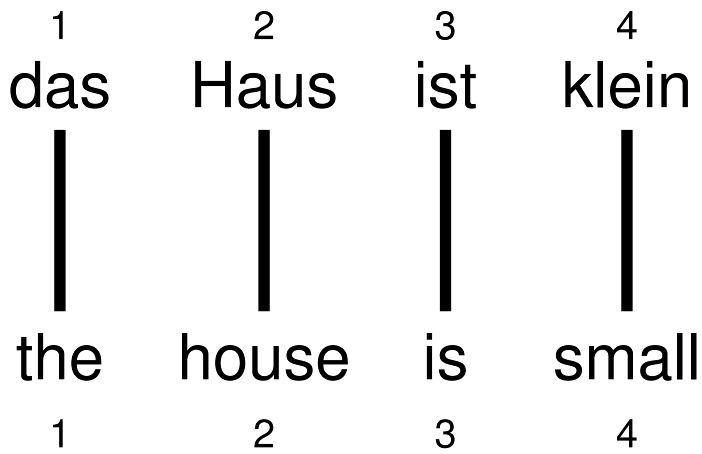
Estimate translation probabilities

- *Maximum likelihood estimation*

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

Alignment

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



- Word *positions* are numbered 1–4

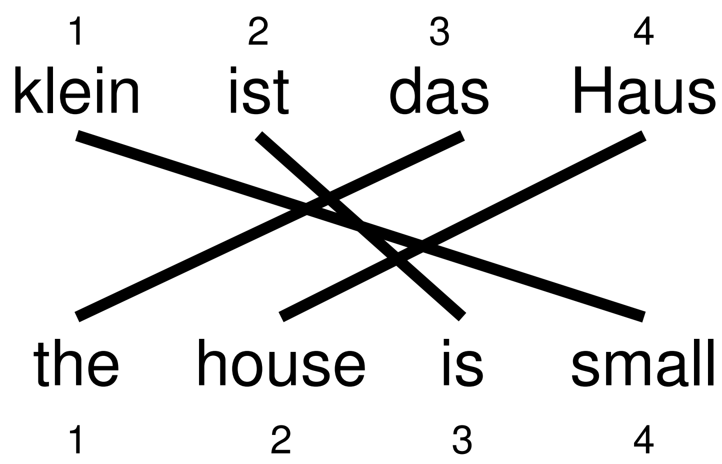
Alignment function

- 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

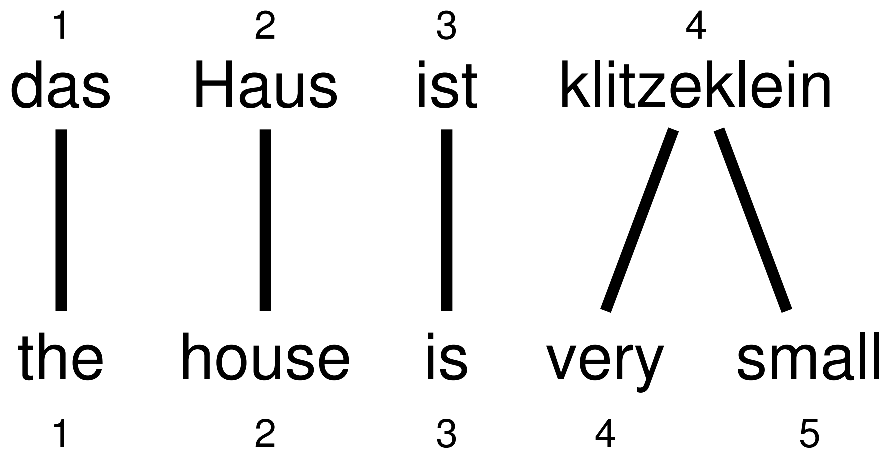
- Words may be **reordered** during translation



$$a : \{1 \rightarrow 3, 2 \rightarrow 4, 3 \rightarrow 2, 4 \rightarrow 1\}$$

One-to-many translation

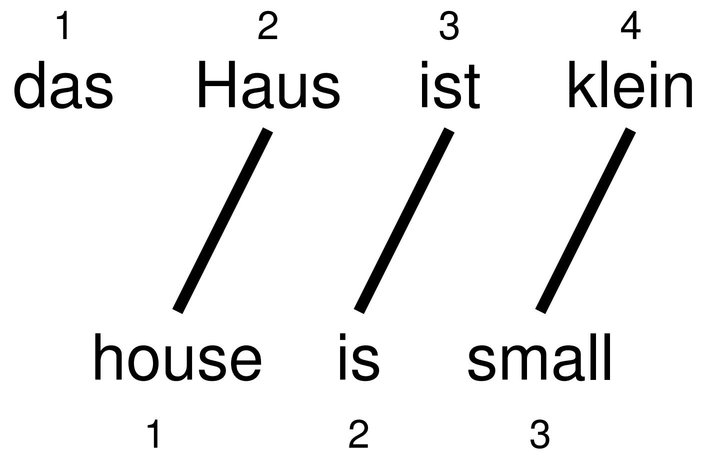
- A source word may translate into **multiple** target words



$$a : \{1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, 4 \rightarrow 4, 5 \rightarrow 4\}$$

Dropping words

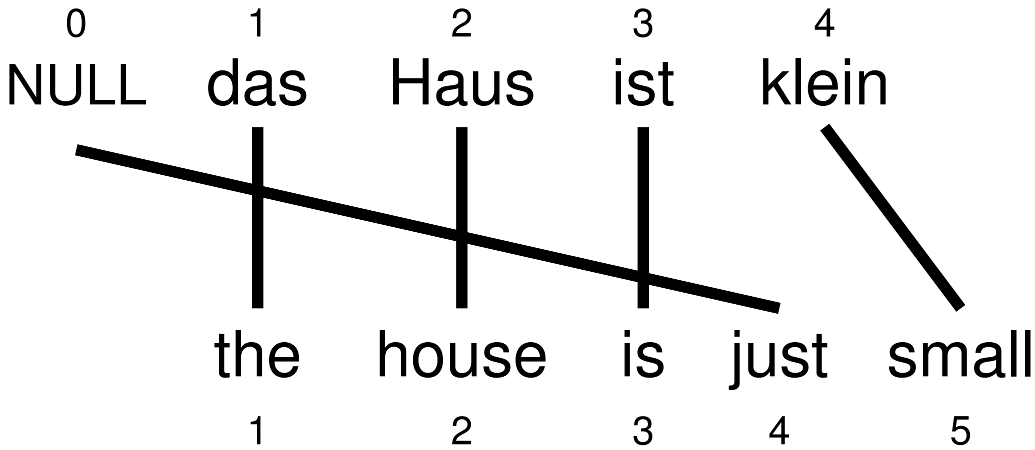
- Words may be **dropped** when translated
  - The German article *das* is dropped



$$a : \{1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4\}$$

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

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

  - parameter ϵ is a *normalization constant*

Example

das

e	$t(e\|f)$
the	0.7
that	0.15
which	0.075
who	0.05
this	0.025

Haus

e	$t(e\|f)$
house	0.8
building	0.16
home	0.02
household	0.015
shell	0.005

ist

e	$t(e\|f)$
is	0.8
's	0.16
exists	0.02
has	0.015
are	0.005

klein

e	$t(e\|f)$
small	0.4
little	0.4
short	0.1
minor	0.06
petty	0.04

$$\begin{aligned} p(e,a|f) &= \frac{\epsilon}{5^4} \times t(\text{the}|\text{das}) \times t(\text{house}|\text{Haus}) \times t(\text{is}|\text{ist}) \times t(\text{small}|\text{klein}) \\ &= \frac{\epsilon}{5^4} \times 0.7 \times 0.8 \times 0.8 \times 0.4 \\ &= 0.0028\epsilon \end{aligned}$$

Learning lexical translation models

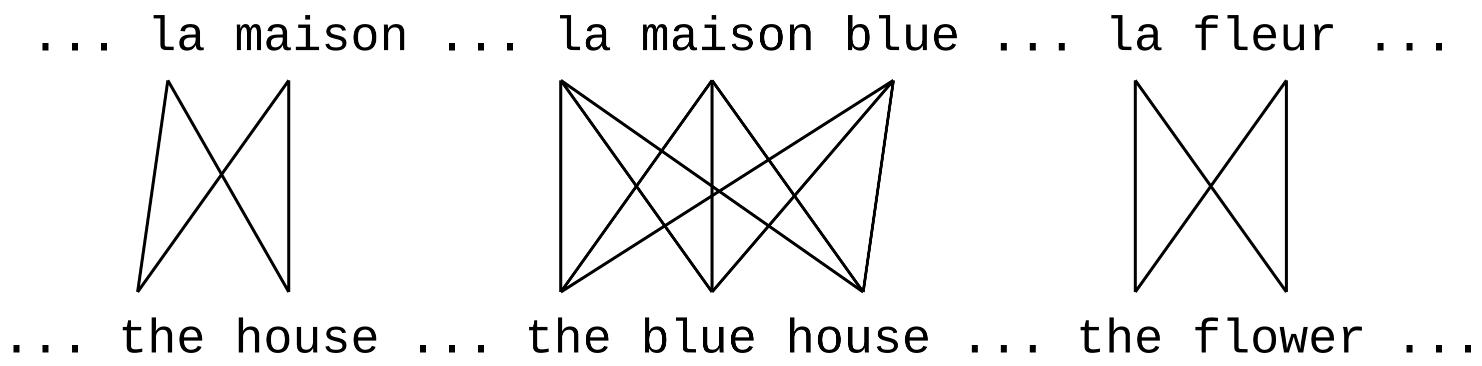
- We would like to *estimate* the lexical translation probabilities $t(e|f)$ from a parallel corpus
- ... but we do not have the alignments
- **Chicken and egg problem**
  - if we had the *alignments*,
    $\rightarrow$ we could estimate the *parameters* of our generative model
  - if we had the *parameters*,
    $\rightarrow$ 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

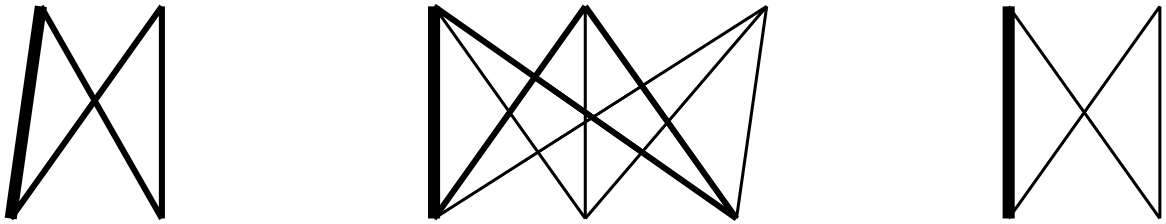
EM algorithm



- 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 ...

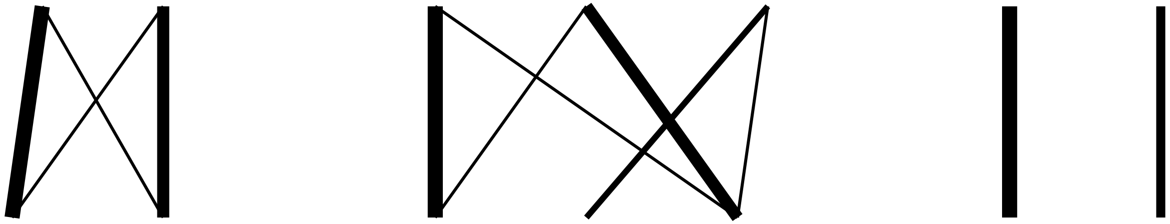


... the house ... the blue house ... the flower ...

- After one iteration
- Alignments, e.g., between *la* and *the* are more likely

EM algorithm

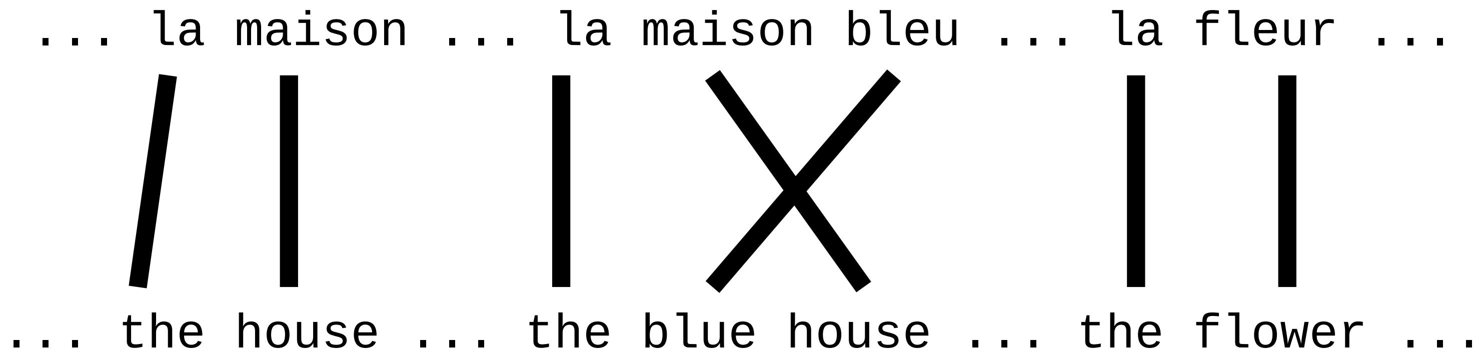
... la maison ... la maison bleu ... la fleur ...



... 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



- Convergence
- Inherent hidden structure revealed by EM

EM algorithm

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

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

```
p(la|the) = 0.453
p(le|the) = 0.334
p(maison|house) = 0.876
p(bleu|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

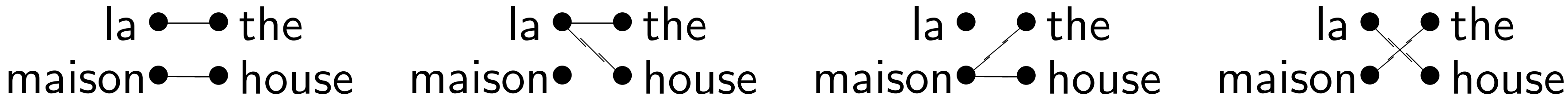
- We need to be able to compute:
  - Expectation-Step: probability of alignments
  - Maximization-Step: count collection

IBM Model 1 and EM

- **Probabilities**

$$p(\text{the}|\text{la}) = 0.7 \qquad p(\text{house}|\text{la}) = 0.05$$
$$p(\text{the}|\text{maison}) = 0.1 \qquad p(\text{house}|\text{maison}) = 0.8$$

- **Alignments**



$$p(\mathbf{e}, a|\mathbf{f}) = 0.56 \qquad p(\mathbf{e}, a|\mathbf{f}) = 0.035 \qquad p(\mathbf{e}, a|\mathbf{f}) = 0.08 \qquad p(\mathbf{e}, a|\mathbf{f}) = 0.005$$

$$p(a|\mathbf{e}, \mathbf{f}) = 0.824 \qquad p(a|\mathbf{e}, \mathbf{f}) = 0.052 \qquad p(a|\mathbf{e}, \mathbf{f}) = 0.118 \qquad p(a|\mathbf{e}, \mathbf{f}) = 0.007$$

- **Counts**

$$c(\text{the}|\text{la}) = 0.824 + 0.052 \qquad c(\text{house}|\text{la}) = 0.052 + 0.007$$
$$c(\text{the}|\text{maison}) = 0.118 + 0.007 \qquad c(\text{house}|\text{maison}) = 0.824 + 0.118$$

IBM Model 1 and EM: Expectation Step

- We need to compute $p(a|\mathbf{e},\mathbf{f})$
- Applying the *chain rule*:

$$p(a|\mathbf{e},\mathbf{f}) = \frac{p(\mathbf{e},a|\mathbf{f})}{p(\mathbf{e}|\mathbf{f})}$$

- We already have the formula for $p(\mathbf{e},\mathbf{a}|\mathbf{f})$ (definition of Model 1)

IBM Model 1 and EM: Expectation Step

- We need to compute $p(\mathbf{e}|\mathbf{f})$

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

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

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

IBM Model 1 and EM: Expectation Step

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

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

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

- Note the trick in the last line
  - removes the need for an *exponential* number of products

$\rightarrow$ this makes IBM Model 1 estimation **tractable**

IBM Model 1 and EM: Expectation Step

- Combine what we have:

$$
\begin{aligned}
p(\mathsf{a}|\mathbf{e},\mathbf{f}) &= p(\mathbf{e},\mathsf{a}|\mathbf{f})/p(\mathbf{e}|\mathbf{f}) \\
&= \frac{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})}{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)} \\
&= \prod_{j=1}^{l_e} \frac{t(e_j|f_{a(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)}
\end{aligned}
$$

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:

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

- With the same simplication as before:

$$c(e|f;\mathbf{e},\mathbf{f}) = \frac{t(e|f)}{\sum_{j=1}^{l_e} t(e|f_{a(j)})} \sum_{j=1}^{l_e} \delta(e,e_j) \sum_{i=0}^{l_f} \delta(f,f_i)$$

IBM Model 1 and EM: Maximization Step

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

$$t(e|f;\mathbf{e},\mathbf{f}) = \frac{\sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f}))}{\sum_f \sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f}))}$$

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

IBM Model 1	lexical translation
IBM Model 2	adds absolute **reordering model**
IBM Model 3	adds **fertility model**
IBM Model 4	relative reordering model
IBM Model 5	fixes **deficiency**

- Only IBM Model 1 has *global maximum*
  - training of a higher IBM model builds on previous model
- Compuationally 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
n(3|slap)
Mary not slap slap slap the green witch
p-null
Mary not slap slap slap NULL the green witch
t(la|the)
Maria no daba una botefada a la verde bruja
d(4|4)
Maria no daba una bofetada a la bruja verde

HMM Model

- Words do not move independently of each other
  - they often move in groups
    $\rightarrow$ condition word movements on previous word
- HMM alignment model:

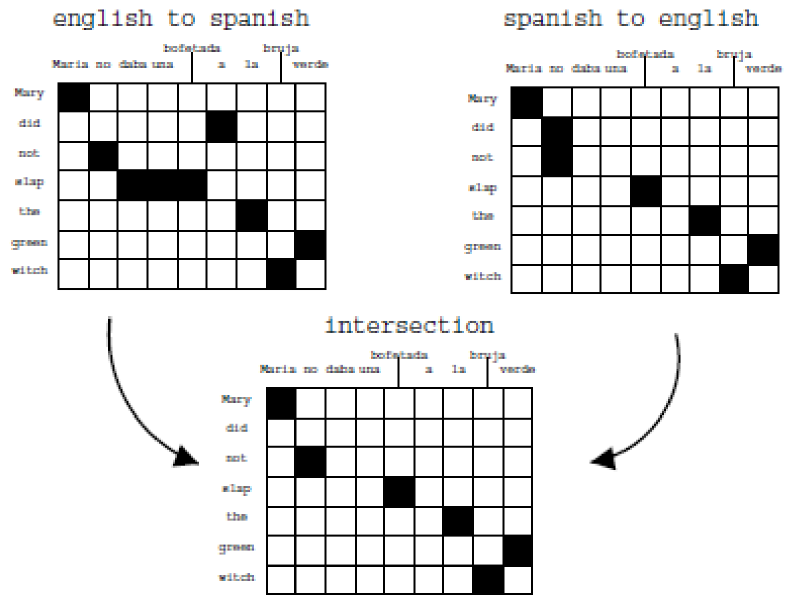
$$p(a(j)|ja(j-1), l_e)$$

- EM algorithm application harder, requires dynamic programming
- IBM Model 4 is similar, also conditions on word classes

Word alignment with IBM models

- IBM Models create a *one-to-many* mapping
  - words are aligned using an alignment function
  - a function may return the same value for different input (one-to-many mapping)
  - a function cannot return multiple values for one input (*no many-to-one mapping*)
- But we need *many-to-many* mappings

Symmetrizing word alignments



- *Intersection* of GIZA++ bidirectional alignments

Symmetrizing word alignments

- *Grow* additional alignment points [Och and Ney, CompLing2003]

Symmetrizing word alignments

```
GROW-DIAG-FINAL(e2f,f2e):
  neighboring = ((-1,0),(0,-1),(1,0),(0,1),(-1,-1),(-1,1),(1,-1),(1,1))
  alignment = intersect(e2f,f2e);
  GROW-DIAG(); FINAL(e2f); FINAL(f2e);
GROW-DIAG():
  iterate until no new points added
    for english word e = 0 ... en
      for foreign word f = 0 ... fn
        if ( e aligned with f )
          for each neighboring point ( e-new, f-new ):
            if ( ( e-new not aligned or f-new not aligned ) and
                ( e-new, f-new ) in union( e2f, f2e ) )
              add alignment point ( e-new, f-new )
FINAL(a):
  for english word e-new = 0 ... en
    for foreign word f-new = 0 ... fn
      if ( ( e-new not aligned or f-new not aligned ) and
           ( e-new, f-new ) in alignment a ) add alignment point ( e-new, f-new )
```

More Recent Work on Symmetrization

- Symmetrize after each iteration of IBM Models [Matusov et al., 2004]
  - run one iteration of E-step for each direction
  - symmetrize the two directions
  - count collection (M-step)

- Use of posterior probabilities in symmetrization
  - generate n-best alignments for each direction
  - calculate how often an alignment point occurs in these alignments
  - use this posterior probability during symmetrization

Discriminative training methods

- Given some annotated training data, supervised learning methods are possible
- Structured prediction
  - not just a classi
    cation problem
  - solution structure has to be constructed in steps
- Many approaches: maximum entropy, neural networks, support vector machines, conditional random fields, MIRA, ...
- Small labeled corpus may be used for parameter tuning of unsupervised aligner [Fraser and Marcu, 2007]

Better Generative Models: Joint Model

$$p(\mathbf{e}, \mathbf{f}) = \sum_{C \in \mathcal{C}} \prod_{<\overline{e}_j, \overline{f}_j> \in C} p(< \overline{e}_j, \overline{f}_j >)$$

- Variables:
  - $\overline{e}_j$ is a phrase in **e**
  - $\overline{f}_j$ is a phrase in **f**
  - C is a set of $< \overline{e}_j, \overline{f}_j >$ which cover all words in **e** and **f**
  - $\mathcal{C}$ is all such sets
- Use EM to estimate $p(< \overline{e}_j, \overline{f}_j >)$ for all phrases in our corpus

Joint Model

- Advantages:
  - Allows phrase-phrase alignments
  - Eliminates need for strange parameters like fertility, NULL alignment
  - Reduces dependency on distortion

- Disadvantages:
  - Complexity explodes - all possible segmentations and their alignments