

# Statistical Machine Translation

## Part II: Word Alignments and EM

**Alexander Fraser**  
CIS, LMU München

2014.10.21 WPCom 1: WSD and MT

# MOOC on Machine Translation

- MT MOOC run by Valencia University starts on October 22nd
- Rule-based, Statistical and Hybrid MT

# Administravia

- It looks like we will start with Referats after the middle of November
- I am thinking about giving out different types of WSD as topics (dictionary-based, supervised, unsupervised, semi-supervised)
  - So far only a small number of people want to do a Referat?
- After that we would discuss Wikification, and integrating WSD into MT
- Question: Is there any interest in projects?

# Where we have been

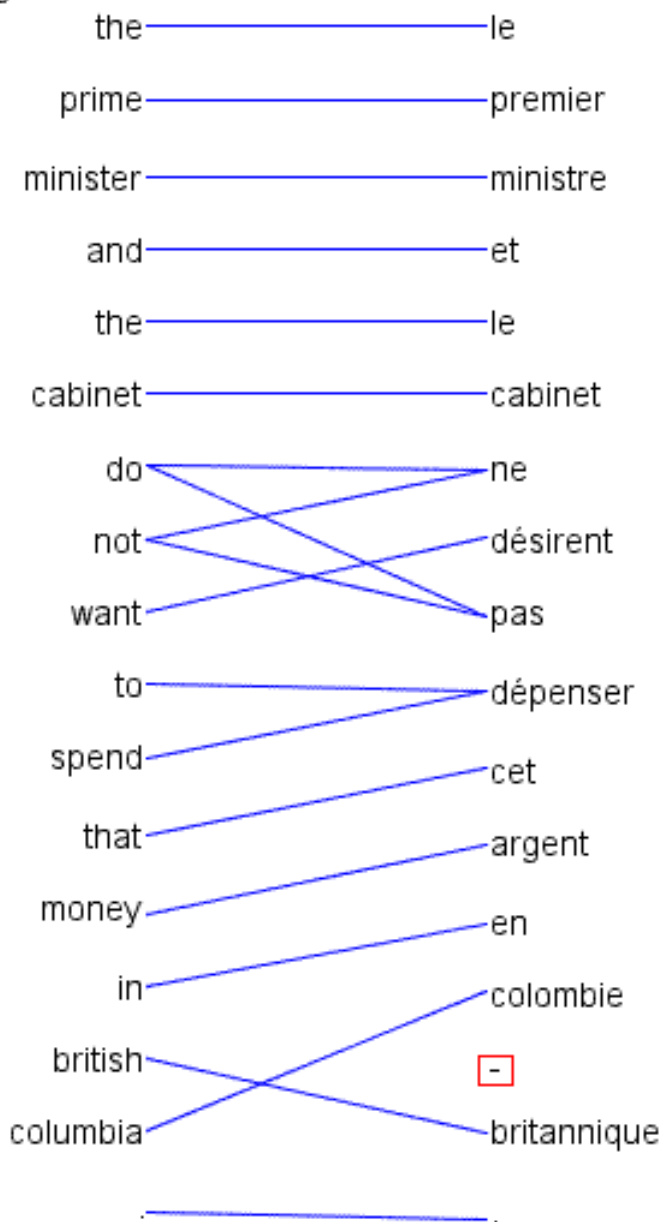
- Parallel corpora
- Sentence alignment
- Overview of statistical machine translation
  - Start with parallel corpus
  - Sentence align it
  - Build SMT system
    - Parameter estimation
  - Given new text, decode
- Human evaluation & BLEU

# Where we are going

- Start with sentence aligned parallel corpus
- Estimate parameters
  - Word alignment
  - Build phrase-based SMT model
- Given new text, translate it!
  - Decoding

# Word Alignments

- Recall that we build translation models from word-aligned parallel sentences
  - The statistics involved in state of the art SMT decoding models are simple
  - Just count translations in the word-aligned parallel sentences
- But what is a word alignment, and how do we obtain it?

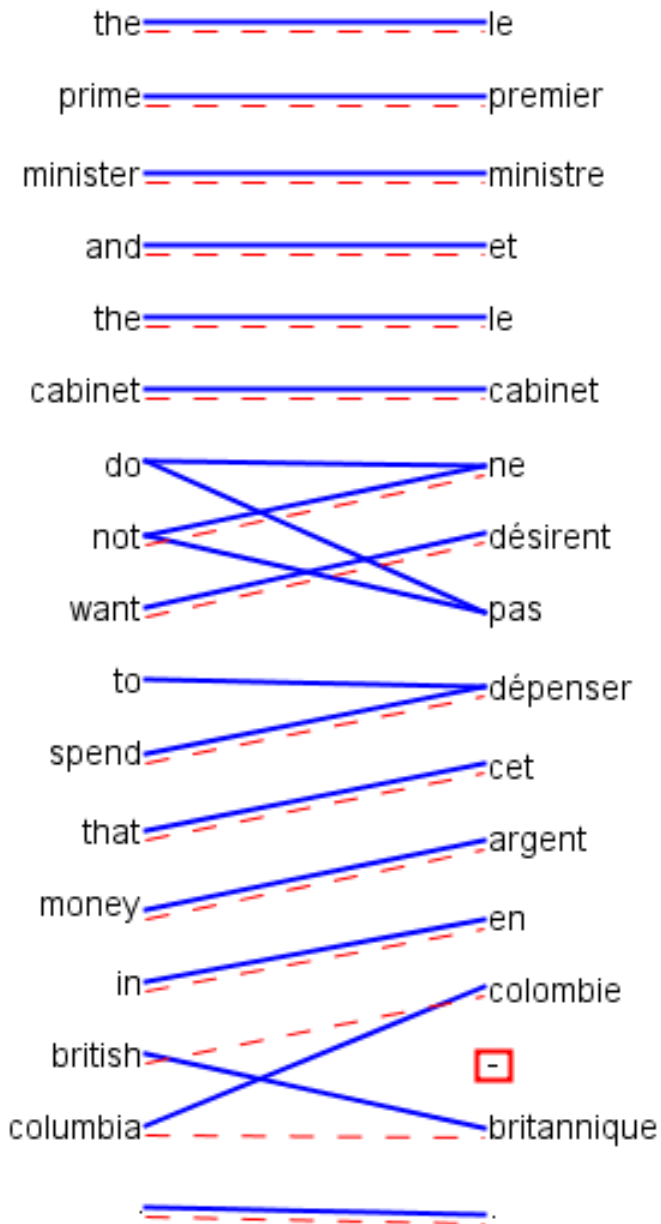


- Word alignment is annotation of minimal translational correspondences

- Annotated in the context in which they occur

- Not idealized translations!

(solid blue lines annotated by a bilingual expert)



- Automatic word alignments are typically generated using a model called IBM Model 4

- No linguistic knowledge

- No correct alignments are supplied to the system

- **Unsupervised learning**

(red dashed line = automatically generated hypothesis)



# Uses of Word Alignment

- Multilingual
  - **Machine Translation**
  - Cross-Lingual Information Retrieval
  - Translingual Coding (Annotation Projection)
  - Document/Sentence Alignment
  - Extraction of Parallel Sentences from Comparable Corpora
- Monolingual
  - Paraphrasing
  - Query Expansion for Monolingual Information Retrieval
  - Summarization
  - Grammar Induction

# Outline

- Measuring alignment quality
- Types of alignments
- IBM Model 1
  - Training IBM Model 1 with Expectation Maximization
- IBM Models 3 and 4
  - Approximate Expectation Maximization
- Heuristics for high quality alignments from the IBM models

# How to measure alignment quality?

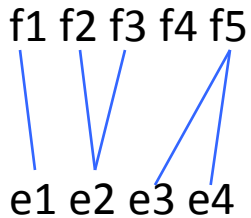
- If we want to compare two word alignment algorithms, we can generate a word alignment with each algorithm for fixed training data
  - Then build an SMT system from each alignment
  - Compare performance of the SMT systems using BLEU
- But this is slow, building SMT systems can take days of computation
  - Question: Can we have an automatic metric like BLEU, but for alignment?
  - Answer: yes, by comparing with gold standard alignments

# Measuring Precision and Recall

- **Precision** is percentage of links in hypothesis that are correct
  - If we hypothesize there are no links, have 100% precision
- **Recall** is percentage of correct links we hypothesized
  - If we hypothesize all possible links, have 100% recall

# $F_\alpha$ -score

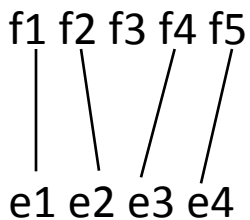
Gold



$$\text{Precision}(A, S) = \frac{|S \cap A|}{|A|} = \frac{3}{4} \quad \begin{array}{l} \text{(e3,f4)} \\ \text{wrong} \end{array}$$

$$\text{Recall}(A, S) = \frac{|S \cap A|}{|S|} = \frac{3}{5} \quad \begin{array}{l} \text{(e2,f3)} \\ \text{(e3,f5)} \\ \text{not in hyp} \end{array}$$

Hypothesis



$$F(A, S, \alpha) = \frac{1}{\frac{\alpha}{\text{Precision}(A, S)} + \frac{1-\alpha}{\text{Recall}(A, S)}}$$

**Called  $F_\alpha$ -score to differentiate from ambiguous term F-Measure**

- Alpha allows trade-off between precision and recall
- But alpha must be set correctly for the task!
- Alpha between 0.1 and 0.4 works well for SMT
  - Biased towards recall

# 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 <i>Haus</i>	Count
<i>house</i>	8,000
<i>building</i>	1,600
<i>home</i>	200
<i>household</i>	150
<i>shell</i>	50



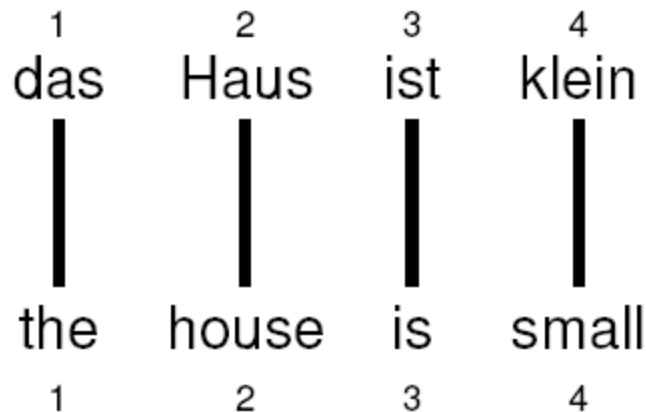
## Estimate translation probabilities

- *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

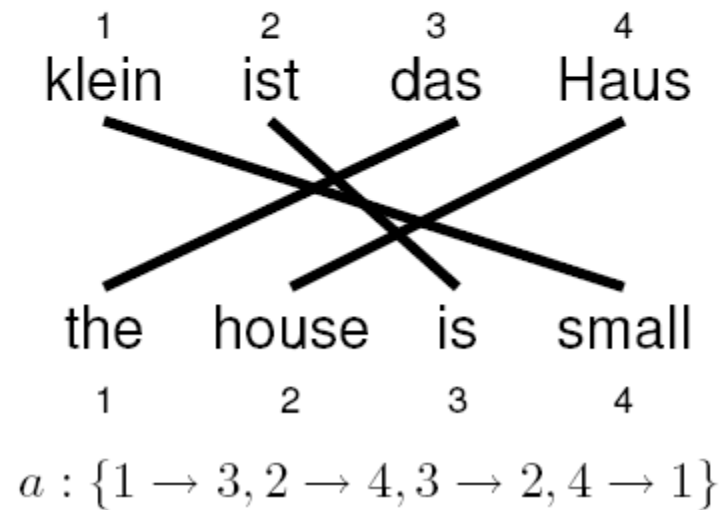
# Alignment function

- 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

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

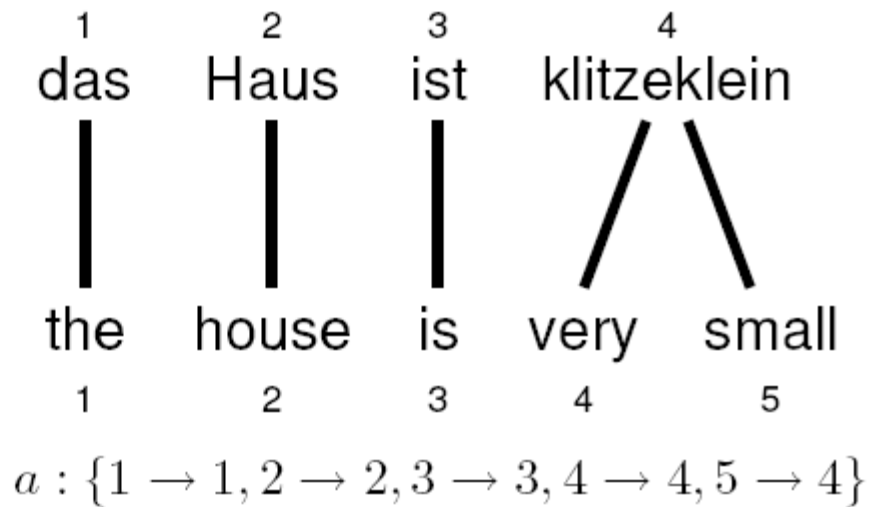
# Reordering

- Words may be **reordered** during translation



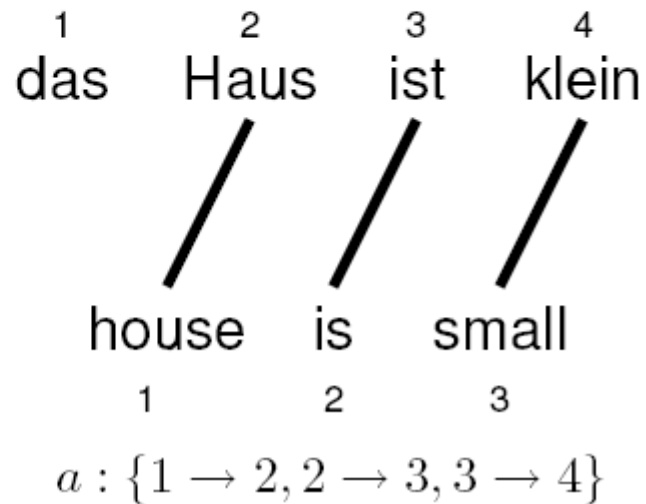
## One-to-many translation

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



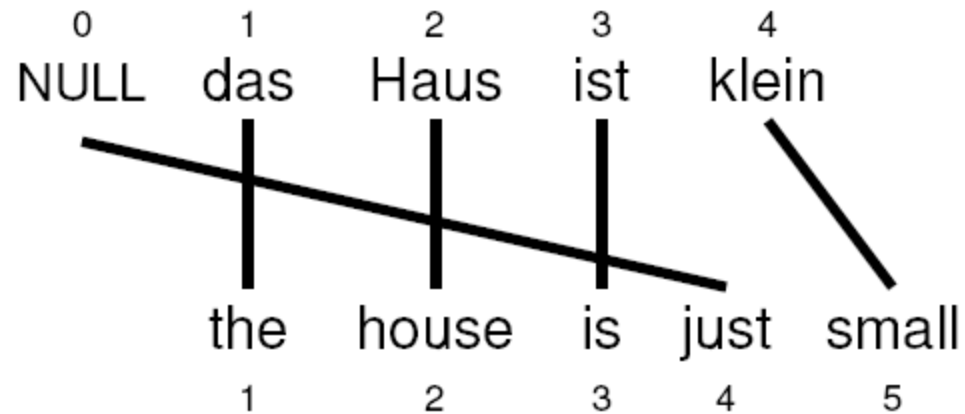
## Dropping words

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



# 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



$$a : \{1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, 4 \rightarrow 0, 5 \rightarrow 4\}$$

# Last word on alignment functions

- Alignments functions are nice because they are a simple representation of the alignment graph
- However, they are strangely asymmetric
  - There is a NULL word on the German side (to explain where unlinked English words came from)
  - But no NULL word on the English side (some German words simply don't generate anything)
  - Very important: alignment functions do not allow us to represent two or more German words being linked to one English word!
    - But we will deal with this later...
- Now let's talk about models



# Generative Word Alignment Models

- We observe a pair of parallel sentences (e,f)
- We would like to know the highest probability alignment **a** for (e,f)
- **Generative** models are models that follow a series of steps
  - We will pretend that e has been generated from f
  - The sequence of steps to do this is encoded in the alignment **a**
  - A generative model associates a probability  $p(e,a | f)$  to each alignment
    - In words, this is the probability of generating the alignment **a** and the English sentence e, given the foreign sentence f

# IBM Model 1

A simple generative model, start with:

- foreign sentence  $f$
- a lexical mapping distribution  $t(\text{EnglishWord} | \text{ForeignWord})$

How to generate an English sentence  $e$  from  $f$ :

1. Pick a length for the English sentence at random
2. Pick an alignment function at random
3. For each English position generate an English word by looking up the aligned ForeignWord in the alignment function, and choose an English word using  $t$

# 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  $\mathbf{f} = (f_1, \dots, f_{l_f})$  of length  $l_f$
  - to an English sentence  $\mathbf{e} = (e_1, \dots, e_{l_e})$  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  $\epsilon$  is a *normalization constant*

## Example

*das*

<i>e</i>	$t(e f)$
<i>the</i>	0.7
<i>that</i>	0.15
<i>which</i>	0.075
<i>who</i>	0.05
<i>this</i>	0.025

*Haus*

<i>e</i>	$t(e f)$
<i>house</i>	0.8
<i>building</i>	0.16
<i>home</i>	0.02
<i>household</i>	0.015
<i>shell</i>	0.005

*ist*

<i>e</i>	$t(e f)$
<i>is</i>	0.8
<i>'s</i>	0.16
<i>exists</i>	0.02
<i>has</i>	0.015
<i>are</i>	0.005

*klein*

<i>e</i>	$t(e f)$
<i>small</i>	0.4
<i>little</i>	0.4
<i>short</i>	0.1
<i>minor</i>	0.06
<i>petty</i>	0.04

$$\begin{aligned}
 p(e,a|f) &= \frac{\epsilon}{5^4} \times t(\text{the}|\text{das}) \times t(\text{house}|\text{Haus}) \times t(\text{is}|\text{ist}) \times t(\text{small}|\text{klein}) \\
 &= \frac{\epsilon}{625} \times 0.7 \quad \times 0.8 \quad \times 0.8 \quad \times 0.4 \\
 &= 0.00029\epsilon
 \end{aligned}$$

# 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*,
    - we could estimate the *parameters* of our generative model
  - if we had the *parameters*,
    - we could estimate the *alignments*

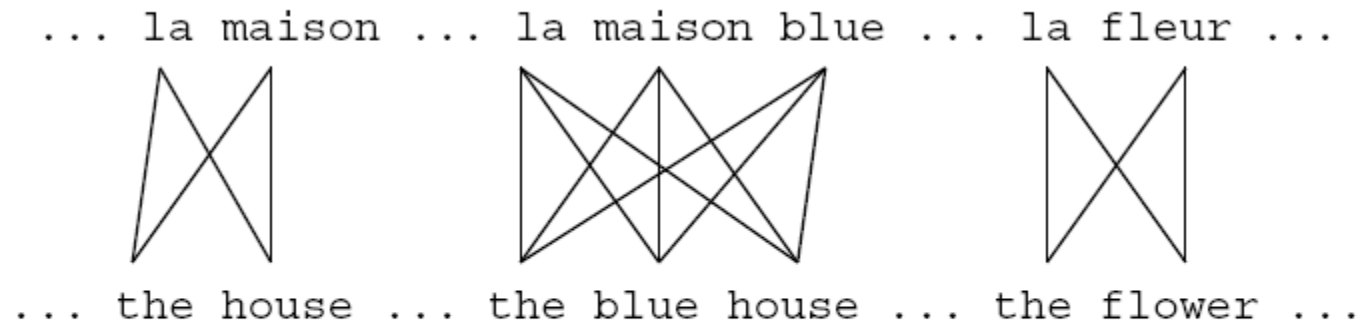
# EM algorithm

- **Incomplete data**
  - if we had *complete data*, we 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

# Unsupervised Training with EM

- Expectation Maximization (EM)
  - Unsupervised learning
  - Maximize the **likelihood** of the training data
    - **Likelihood** is (informally) the probability the model assigns to the training data (pairs of sentences)
  - E-Step: predict according to current parameters
  - M-Step: reestimate parameters from predictions
  - Amazing but true: if we iterate E and M steps, we **increase likelihood\*!**
    - (\*actually, we **do not decrease likelihood**)

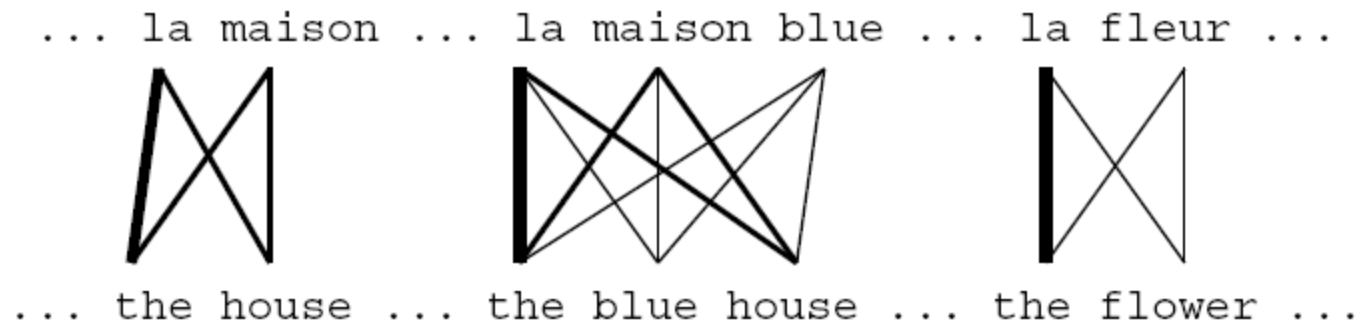
## EM algorithm



- Initial step: all alignments equally likely
- Model learns that, e.g., *la* is often aligned with *the*

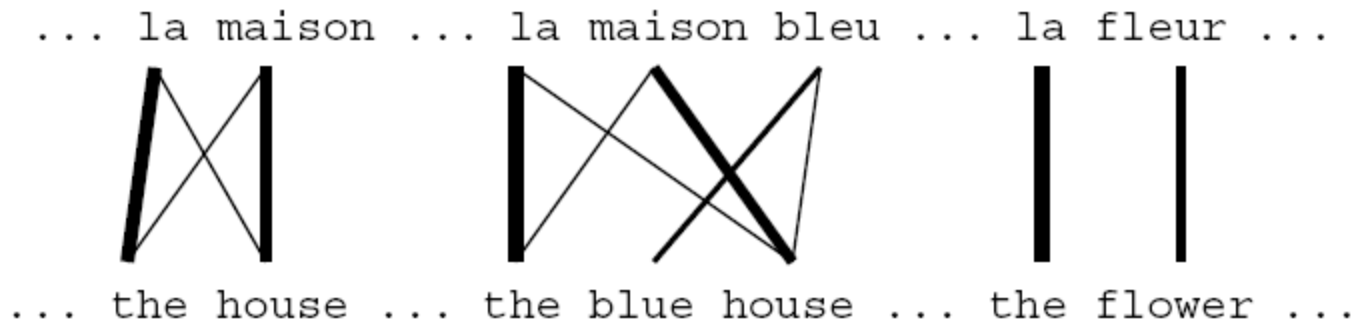


## EM algorithm



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

## EM algorithm



- 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 ...  
/ | | X | |  
... the house ... the blue house ... the flower ...

- Convergence
- Inherent hidden structure revealed by EM

# EM algorithm

... la maison ... la maison bleu ... la fleur ...  
/ | | | X | |  
... the house ... the blue house ... the flower ...



$p(\text{la}|\text{the}) = 0.453$   
 $p(\text{le}|\text{the}) = 0.334$   
 $p(\text{maison}|\text{house}) = 0.876$   
 $p(\text{bleu}|\text{blue}) = 0.563$   
...

- 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 data 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

We will work out an example for the sentence pair:

la maison

the house

in a few slides, but first, let's discuss EM further...

# Implementing the Expectation-Step

- We are given the “t” parameters
- **For each sentence pair:**
- For every possible alignment of this sentence pair, simply work out the equation of Model 1
  - We will actually use the probability of every possible alignment (not just the best alignment!)
- We are interested in the “**posterior probability**” of each alignment
  - We sum the Model 1 alignment scores, over all alignments of a sentence pair
  - Then we will divide the alignment score of each alignment by this sum to obtain a normalized score
    - Note that this means we can ignore the left part of the Model 1 formula, because it is constant over all alignments of a fixed sentence pair
  - The resulting normalized score is the posterior probability of the alignment
    - Note that the sum over the alignments of a particular sentence pair is 1
- The **posterior probability** of each alignment of each sentence pair will be used in the Maximization-Step



# Implementing the Maximization-Step

- For every alignment of every sentence pair we assign weighted counts to the translations indicated by the alignment
  - These counts are weighted by the posterior probability of the alignment
  - Example: if we have many different alignments of a particular sentence pair, and the first alignment has a posterior probability of 0.32, then we assign a “fractional count” of 0.32 to each of the links that occur in this alignment
- Then we collect these counts and sum them over the entire corpus, giving us a **list of fractional counts over the entire corpus**
  - These could, for example, look like:  $c(\text{the} | \text{la}) = 8.0$ ,  $c(\text{house} | \text{la}) = 0.1$ , ...
- Finally we **normalize the counts to sum to 1** for the right hand side of each  $t$  parameter so that we have a conditional probability distribution
  - If the total counts for “la” on the right hand side = 10.0, then, in our example:
  - $p(\text{the} | \text{la}) = 8.0 / 10.0 = 0.80$
  - $p(\text{house} | \text{la}) = 0.1 / 10.0 = 0.01$
  - ...
- These normalized counts are our **new  $t$  parameters!**

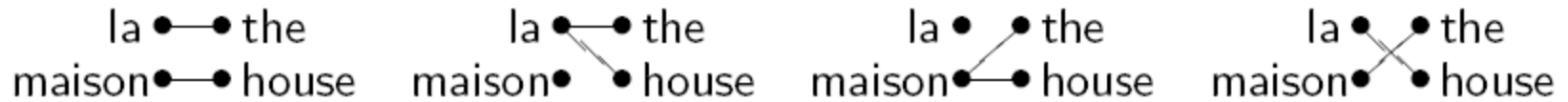
- In the next slide, I will show how to get the fractional counts for our example sentence
  - We do not consider the NULL word
    - This is just to reduce the total number of alignments we have to consider
  - We assume we are somewhere in the middle of EM, not at the beginning of EM
    - This is only because having all  $t$  parameters being uniform would make the example difficult to understand
  - The variable  $z$  is the left part of the Model 1 formula
    - This term is the same for each alignment, so it cancels out when calculating the posterior!

# IBM Model 1 and EM

- **Probabilities**

$$\begin{aligned}
 p(\text{the}|\text{la}) &= 0.7 & p(\text{house}|\text{la}) &= 0.05 \\
 p(\text{the}|\text{maison}) &= 0.1 & p(\text{house}|\text{maison}) &= 0.8
 \end{aligned}$$

- **Alignments**



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

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

- **Counts**

$$\begin{aligned}
 c(\text{the}|\text{la}) &= 0.824 + 0.052 & c(\text{house}|\text{la}) &= 0.052 + 0.007 \\
 c(\text{the}|\text{maison}) &= 0.118 + 0.007 & c(\text{house}|\text{maison}) &= 0.824 + 0.118
 \end{aligned}$$

# More formal and faster implementatation: EM for Model 1

- If you understood the previous slide, you understand EM training of Model 1
- However, if you implement it this way, it will be **slow** because of the enumeration of all alignments
- The next slides show:
  1. A more mathematical presentation with the foreign NULL word included
  2. A trick which allows a very efficient (and incredibly simple!) implementation
    - We will be able to completely avoid enumerating alignments and directly obtain the counts we need!

## 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  $p(\mathbf{e}|\mathbf{f})$

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

## IBM Model 1 and EM: Expectation Step

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

- Note the trick in the last line
  - removes the need for an *exponential* number of products
  - this makes IBM Model 1 estimation **tractable**

## The trick

(case  $l_e = l_f = 2$ )

$$\sum_{a(1)=0}^2 \sum_{a(2)=0}^2 = \frac{\epsilon}{3^2} \prod_{j=1}^2 t(e_j | f_{a(j)}) =$$

$$\begin{aligned} &= t(e_1/f_0) t(e_2/f_0) + t(e_1/f_0) t(e_2/f_1) + t(e_1/f_0) t(e_2/f_2) \\ &\quad + t(e_1/f_1) t(e_2/f_0) + t(e_1/f_1) t(e_2/f_1) + t(e_1/f_1) t(e_2/f_2) \\ &\quad + t(e_1/f_2) t(e_2/f_0) + t(e_1/f_2) t(e_2/f_1) + t(e_1/f_2) t(e_2/f_2) \end{aligned}$$

$$\begin{aligned} &= t(e_1/f_0) [t(e_2/f_0) + t(e_2/f_1) + t(e_2/f_2)] \\ &\quad + t(e_1/f_1) [t(e_2/f_0) + t(e_2/f_1) + t(e_2/f_2)] \\ &\quad + t(e_1/f_2) [t(e_2/f_0) + t(e_2/f_1) + t(e_2/f_2)] \end{aligned}$$

$$= [t(e_1/f_0) + t(e_1/f_1) + t(e_1/f_2)] [t(e_2/f_0) + t(e_2/f_1) + t(e_2/f_2)]$$



## IBM Model 1 and EM: Expectation Step

- Combine what we have:

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

# Collecting Counts

- We now have to collect counts from each sentence pair  $\mathbf{e}$ ,  $\mathbf{f}$  for each word pair  $e$  and  $f$
- The formula for fixed words  $e$  and  $f$  is on the next slide
- We first need the definition of the Kronecker delta function:

$$\delta(a,b) = 1 \text{ if } a=b$$

0 otherwise

# IBM Model 1 and EM: Maximization Step

- Now we have to *collect counts*
- Evidence from a sentence pair  $\mathbf{e}, \mathbf{f}$  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 simplification as before:

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

# IBM Model 1 and EM: Maximization Step

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

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

# IBM Model 1 and EM: Pseudocode

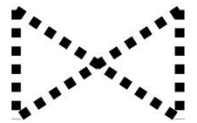
**Input:** set of sentence pairs  $(\mathbf{e}, \mathbf{f})$


**Output:** translation prob.  $t(e|f)$


```
1: initialize  $t(e|f)$  uniformly
2: while not converged do
3:   // initialize
4:    $\text{count}(e|f) = 0$  for all  $e, f$ 
5:    $\text{total}(f) = 0$  for all  $f$ 
6:   for all sentence pairs  $(\mathbf{e}, \mathbf{f})$  do
7:     // compute normalization
8:     for all words  $e$  in  $\mathbf{e}$  do
9:        $\text{s-total}(e) = 0$ 
10:      for all words  $f$  in  $\mathbf{f}$  do
11:         $\text{s-total}(e) += t(e|f)$ 
12:      end for
13:    end for
```

```
14:   // collect counts
15:   for all words  $e$  in  $\mathbf{e}$  do
16:     for all words  $f$  in  $\mathbf{f}$  do
17:        $\text{count}(e|f) += \frac{t(e|f)}{\text{s-total}(e)}$ 
18:        $\text{total}(f) += \frac{t(e|f)}{\text{s-total}(e)}$ 
19:     end for
20:   end for
21: end for
22: // estimate probabilities
23: for all foreign words  $f$  do
24:   for all English words  $e$  do
25:      $t(e|f) = \frac{\text{count}(e|f)}{\text{total}(f)}$ 
26:   end for
27: end for
28: end while
```

# Convergence

das Haus  
  
 the house

das Buch  
  
 the book

ein Buch  
  
 a book

$e$	$f$	initial	1st it.	2nd it.	3rd it.	...	final
the	das	0.25	0.5	0.6364	0.7479	...	1
book	das	0.25	0.25	0.1818	0.1208	...	0
house	das	0.25	0.25	0.1818	0.1313	...	0
the	buch	0.25	0.25	0.1818	0.1208	...	0
book	buch	0.25	0.5	0.6364	0.7479	...	1
a	buch	0.25	0.25	0.1818	0.1313	...	0
book	ein	0.25	0.5	0.4286	0.3466	...	0
a	ein	0.25	0.5	0.5714	0.6534	...	1
the	haus	0.25	0.5	0.4286	0.3466	...	0
house	haus	0.25	0.5	0.5714	0.6534	...	1

## Higher IBM Models

IBM Model 1	lexical translation
IBM Model 2	adds absolute <b>reordering model</b>
IBM Model 3	adds <b>fertility model</b>
IBM Model 4	relative reordering model
IBM Model 5	fixes <b>deficiency</b>

- 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

- Thank you for your attention!