Alignment Algorithms

UNIVERSITÄT DES SAARLANDES

Andreas Eisele
DFKI GmbH, LT Lab
eisele@dfki.de

Introduction to Computational Linguistics, SS 2010

Alignment Algorithms

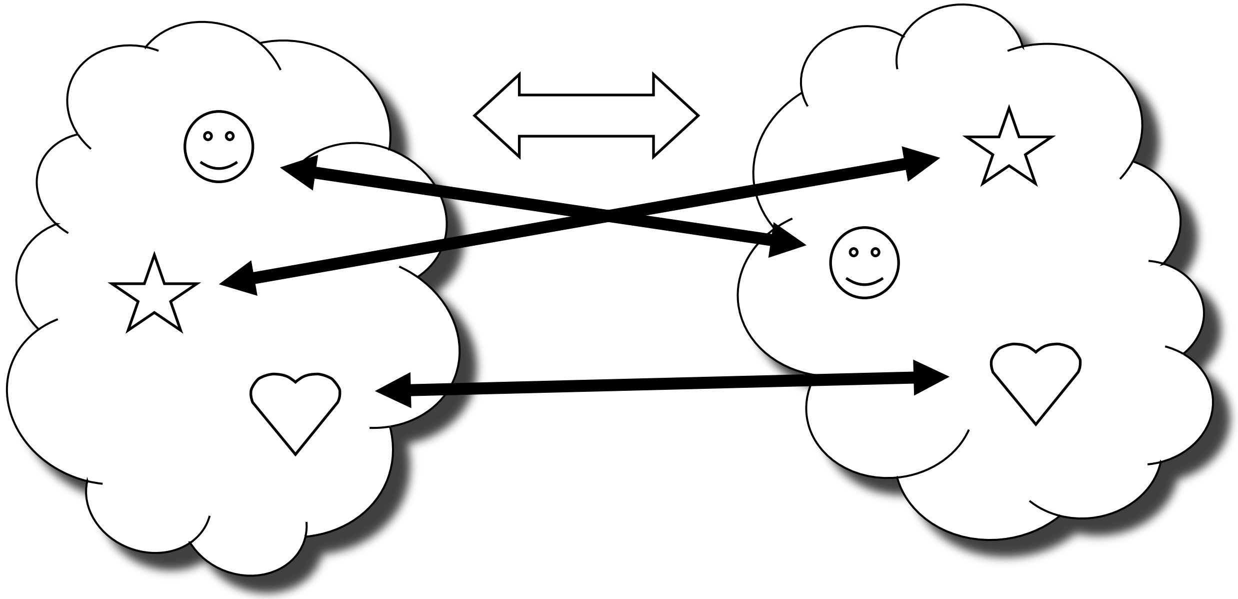
UNIVERSITÄT DES SAARLANDES

today's plan:

- Motivation: What is alignment? What is it good for? What forms can it take?
- How to find longest common subsequences (LCS)
  - Simple application of "dynamic programming"
- Sentence alignment in bilingual documents
  - Same approach, but more sophisticated
- A simplified model for word alignment
  - Which words in translated texts correspond?

What is alignment?

Given two or more structures that have corresponding parts, find out the correspondences



What is alignment good for?

On an abstract level, there are four broad reasons to compute alignments of structures:

- Judging the similarity of the structures
- Extracting the corresponding parts
- Merging into larger structure
- Surgery: replacing some of the parts

How difficult is alignment?

UNIVERSITÄT DES SAARLANDES

Difficulty of alignment ranges from fairly easy to arbitrarily complex.
There are several aspects that may contribute to the difficulty

- Similarity metric to judge whether parts correspond
  - corresponding parts are equal
  - corresponding parts are similar wrt. given metric
  - similarity metric needs to be induced from data
  - alignment on a finer level (recursive alignment)
- Properties of structures
  - sequences
  - trees
  - unordered collections
  - others…
- Size of the structures
  - larger structures lead to larger search spaces, are more difficult to align
- Constraints on alignments, e.g. monotonicity, exhaustivity, structural similarity
  - stronger constraints can make search faster, but may also lead to more complex algorithms
  - non-exhaustive alignment is easier to compute, but less informative (➔ bootstrapping less effective)

Applications of Alignment in NLP

Only a small sample of all the possibilities…

Structures	Parts to align	What to do	What for
Files	**Lines**	**Find (minimal set of) differences**	**general-purpose tool diff, version control (CVS, SVN)**
Word Sequences	**Words**	**Textual similarity**	**Document retrieval, clustering, plagiarism detection, MT evaluation**
Words/ phoneme strings	**characters/phonemes**	**phonetic alignment**	**Speech recognition, speech synthesis, phonetic search**
Character Strings (Words)	**Characters**	**Longest common subsequence, similar words wrt. given metric**	**Spelling correction, post-OCR/ post-ASR correction**
Text + Translation	**Sentences**	**Align sentences with their translation**	**Translation memories, statistical and example-based machine translation**
Sentence + Translation	**Words/phrases**	**Align words and phrases**	**Statistical machine translation**
Syntactic structures	**Constituents**	**Align corresponding constituents**	**Adaptive MT, translate between annotation formalisms**
HTML code	**relevant text pieces**	**find out what is relevant**	**wrapper induction for building crawlers, IE systems**

Applications other than NLP

…are numerous, such as

- Bio-Informatics:
  - Find genes, judge similarities between species, reconstruct historical evolution
  - Alignment between different representations, e.g. DNA and protein sequences
- Data mining in general
- Fusing output of multiple sensors
  - identification of objects in stereo vision
  - fusing multiple (audio-, video-) channels from meeting recordings

Longest Common Subsequences

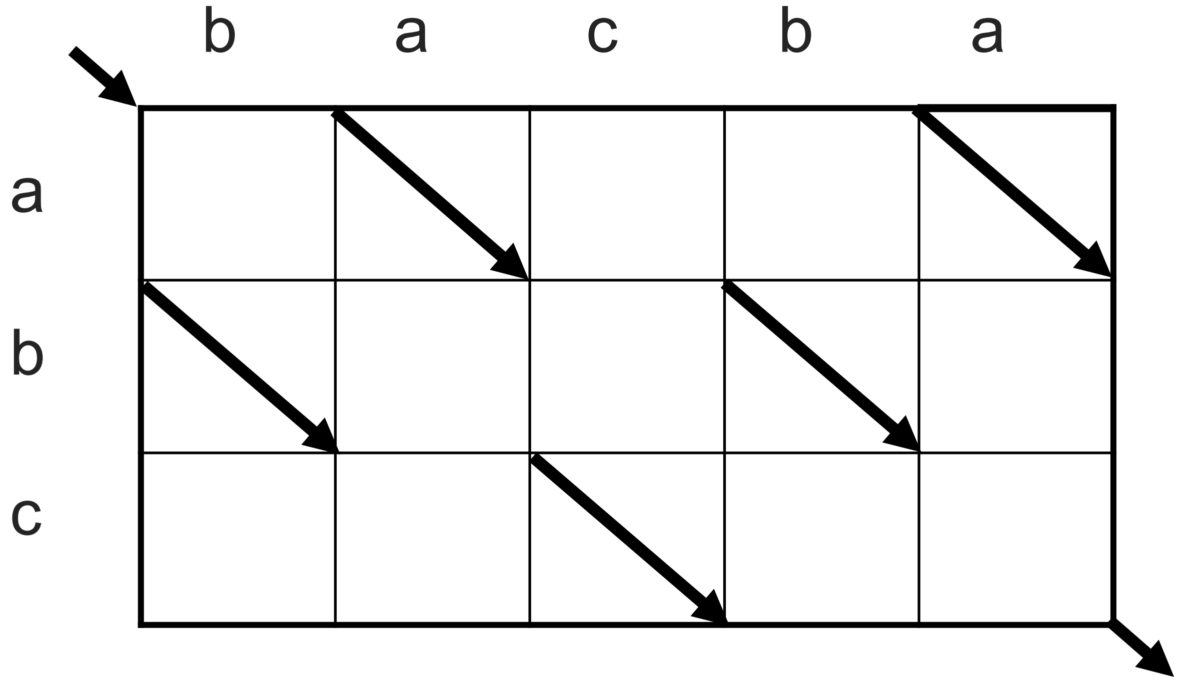
- Given strings s_1 and s_2, the goal is to find a common subsequence of maximal length (LCS)
- Finding LCSs constitutes the simplest form of alignment between strings, but has very many uses in NLP, such as measuring similarities of words, sentences, documents, phoneme sequences, …
- Several popular notions in text processing are closely related, such as Levenshtein distance, edit-distance
- A string s has $2^{|s|}$ subsequences, hence an algorithm based on a naïve enumeration would be far too expensive
- Better: decompose into smaller problems, solve them in a systematic order and avoid repeated computation of identical sub-problem, i.e. apply

  “dynamic programming (DP)”

Find LCS via Dynamic Programming

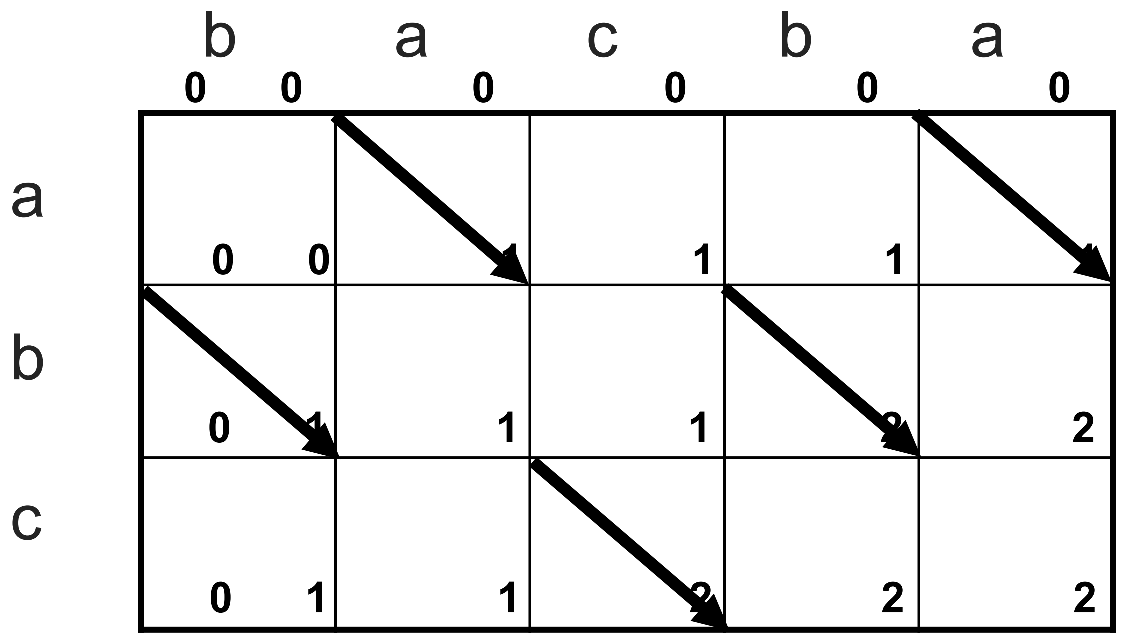
Assume we search LCS("abc", "bacba")

We can draw a rectangular graph of all possible character alignments



Find LCS via Dynamic Programming

We can traverse the rectangle and compute the length of the LCS of the corresponding prefixes



Find LCS via Dynamic Programming

UNIVERSITÄT DES SAARLANDES

The idea is realized in the following little algorithm (expressed in Python 2.X for concreteness):

```
def matrix(l1,l2): return [[0 for i2 in range(l2)] for i1 in range(l1)]

def match(c1,c2): return 0+(c1==c2)

def lcsScores(s1,s2):
    l1,l2 = len(s1),len(s2)
    score = matrix(l1+1,l2+1)
    for i1 in range(l1):
       for i2 in range(l2):
            score[i1+1][i2+1] = \
                max(score[i1][i2+1],
                    score[i1+1][i2],
                    score[i1][i2]+match(s1[i1],s2[i2]))
    return score

def lenLCS(s1, s2): return lcsScores(s1, s2)[-1][-1]
```

Find LCS via Dynamic Programming

In order to actually extract one of the longest subsequences, one can traverse the array backwards

```
def lcs(s1,s2):
    score = lcsScores(s1,s2)
    i1 = len(s1)
    i2 = len(s2)
    res = []
    while score[i1][i2]:
        if s1[i1-1]==s2[i2-1]:
            i1 -= 1 ; i2 -= 1
            res.insert(0,s1[i1])
        elif score[i1-1][i2] > score[i1][i2-1]: i1 -= 1
        else: i2 -= 1
    return res
```

Find LCS via Dynamic Programming

The function lcs() does not explicitly state the alignment between the two sequences. To cure this, we can e.g. insert explicit traces of null-alignments into the result (and obtain pairs of results with lengths equal to the input arguments):

```
def lcsTrace(s1,s2):
    score = lcsScores(s1,s2)
    i1 = len(s1)
    i2 = len(s2)
    res1 = []; res2 = []
    while score[i1][i2]:
        if s1[i1-1]==s2[i2-1]:
            i1 -= 1 ; i2 -= 1
            res1.insert(0,s1[i1])
            res2.insert(0,s1[i1])
        elif score[i1-1][i2] > score[i1][i2-1]:
            i1 -= 1
            res1.insert(0,None)
        else:
            i2 -= 1
            res2.insert(0,None)
    for i in range(i1): res1.insert(0,None)
    for i in range(i2): res2.insert(0,None)

    return res1, res2
```

Sentence Alignment

UNIVERSITÄT DES SAARLANDES

The problem:

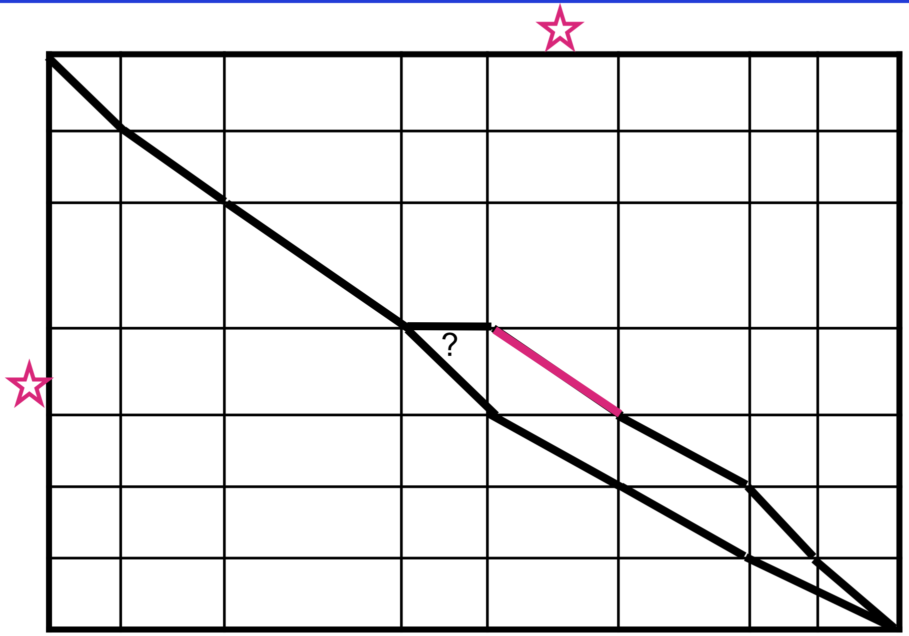
- Assume you are given an English text of 100 sentences and a German translation consisting of 102 sentences, and you want to find corresponding sentence pairs
- Closer inspection shows that some sentences are omitted, sometimes sentences are split or merged or swapped.
- Assume you need to automate the process of aligning the sentences (e.g. as a module in a translation memory system)

Sentence Alignment

The solution:

- The basic structure of a possible algorithm corresponds exactly the DP computation of the LCS (the details are messier…)
- Useful similarity measures:
  - Church/Gale: estimate probabilities for n-m matches from sample, estimate probabilities that sentences (groups) of given length correspond
  - Kay/Röscheisen: count pairs of words that are known to be translations, bootstrap alignments on sentence and word level
  - Even simpler: maximize the overall text length that can be explained by the alignment

Sentence alignment



Complexity is O(n*m)

Additional evidence (e.g. from invariant or cognate words) can be helpful

Sentence Alignment

UNIVERSITÄT DES SAARLANDES

Code used in a real project (not optimized for efficiency):

```
bonus=80
alignmentPatterns = [(1, 1, bonus), (0, 1, 1), (1, 0, 1), (2, 2, bonus)] + \
                    [(1, o, bonus) for o in range(2,13)] + \
                    [(o, 1, bonus) for o in range(2,13)]

def sentAlign(sentsX, sentsY, alignmentPatterns, separator="§"):
    matrix=[]
    for x in range(len(sentsX)+1):
        row=[]
        matrix+=[row]
        for y in range(len(sentsY)+1):
            paths=[(-1,[],[])]
            for (dX, dY, bonus) in alignmentPatterns:
                (sX,sY) = (x-dX, y-dY)
                if min(sX,sY) >= 0 :
                    (sScore,sslX, sslY)=matrix[sX][sY]
                    incX=separator.join(sentsX[sX:x])
                    incY=separator.join(sentsY[sY:y])
                    incScore=min(len(incX),len(incY))
                    bonus=min(bonus,incScore)
                    paths+= [(sScore+incScore+bonus,sslX+[incX], sslY+[incY])]
            paths.sort()
            row+=[paths[-1]]
    return matrix[-1][-1]
```

Sentence Alignment in Practice

Commercially useful sentence alignment algorithms need to be highly sophisticated, because:

- Real data is messy, so algorithms need to be robust
- Multiple knowledge sources need to be integrated (cognates, heuristics, lexicons, …)
- Texts may have hierarchical structure that can/should be exploited
- Texts can be long, so a simple O(n*m) algorithm may consume too much space/time ➔ need beam search
- Users would like to see what is going on, and also need possibility to correct the results
- Sentence alignment and word alignment should feed each other in a bootstrapping process

Word alignment

The problem:

- We need to know alignments between texts and translations on word or phrase level (useful for terminology extraction, statistical MT, improved translation memories, algorithms for translation checking etc.)
- This is more difficult as for sentences, as the order on both sides does not agree
- There is no a-priory notion of similarity, possible correspondences need to be learned from data

Word alignment

A typical solution

- Assume a probabilistic model for co-occurrences between words/phrases
- Train parameters from data

But we have a chicken-and-egg situation:

- given alignments, we can learn the parameters
- given parameters, we can estimate alignments
- we don’t know how to start

Expectation-maximization (EM)

- Similar situations are ubiquitous in learning stochastic models from raw data lacking annotation (NLP: HMMs, PCFGs, …)
- There is a generic scheme for how to deal with this problem, called EM algorithm
- Basic idea:
  - Start with a simple model (e.g. a uniform probability distribution)
  - Estimate a probabilistic annotation
  - Train a model from this estimate
  - Iterate re-estimation until result is good enough
- Properties of EM:
  - Likelihood of model is guaranteed to increase in each iteration
  - EM hence converges towards a maximum likelihood estimate (MLE)
  - But this maximum is only local
  - (Even global) MLE need not be useful for unseen data, less iterations may give better models

Simplified model for word alignment

- Researchers at IBM have developed a cascade of approaches, called IBM Models 1…5 for statistical MT
- In the sequel, we will use a simplified version of IBM Model 1 (called Model 0), assuming that each word in a foreign language text f is the translation of (generated by) some word in the English version e
- Probability that the *i*th foreign word f_i is generated, given an English sentence e, is modeled as:

$$P(f_i|e) = \sum_j P(f_i \mid e_j)$$

- Probability that the complete foreign sentence is generated (omitting some boring details):

$$P(f|e) = \prod_i P(f_i|e) = \prod_i \sum_j P(f_i \mid e_j)$$

EM algorithm for word alignment

- From a set of annotated data (i.e. sentence pairs with word alignments), we can obtain a new translation model:

$$P(f_i|e_j) = freq(f_i,e_j) / freq(e_j)$$

- From a model P, a foreign word f_i, and a sequence $e = e_1 \ldots e_n$ of possible “causes”, we can estimate frequencies as

$$freq(f_i|e_j) = P(f_i|e_j) / \sum_{k=1}^{n} P(f_i|e_k)$$

The training corpus and models

Corpus:

chien méchant ⇔ dangerous dog

petit chien ⇔ small dog

Initial model:

$p_0(f_i|e_j) = \text{constant}$

Update steps:

see last slide

EM iteration 1

UNIVERSITÄT DES SAARLANDES

Local frequency estimates

$freq(f_i\|e_j)$	chien	méchant
dangerous	0.5	0.5
dog	0.5	0.5

$freq(f_i\|e_j)$	petit	chien
small	0.5	0.5
dog	0.5	0.5

Global frequencies and probabilities

$freq(f_i\|e_j)$	petit	chien	méchant
small	0.5	0.5	
dangerous		0.5	0.5
dog	0.5	1.0	0.5

$p(f_i\|e_j)$	petit	chien	méchant
small	0.5	0.5	
dangerous		0.5	0.5
dog	0.25	0.5	0.25

EM iteration 2

Probabilities from iteration 1

$p(f_i\|e_j)$	petit	chien	méchant
small	0.5	0.5	
dangerous		0.5	0.5
dog	0.25	0.5	0.25

New frequency estimates

$freq(f_i\|e_j)$	chien	méchant
dangerous	0.5	0.67
dog	0.5	0.33

$freq(f_i\|e_j)$	petit	chien
small	0.67	0.5
dog	0.33	0.5

EM iteration 2

Local frequency estimates

$freq(f_i \mid e_j)$	chien	méchant
dangerous	0.5	0.67
dog	0.5	0.33

$freq(f_i \mid e_j)$	petit	chien
small	0.67	0.5
dog	0.33	0.5

Global frequencies and probabilities

$freq(f_i \mid e_j)$	petit	chien	méchant
small	0.67	0.5	
dangerous		0.5	0.67
dog	0.33	1.0	0.33

$p(f_i \mid e_j)$	petit	chien	méchant
small	0.57	0.43	
dangerous		0.43	0.57
dog	0.2	0.6	0.2

IBM Model I

Idea:

- Each word of the foreign sentence is generated/ explained by some English word
- There is no limitation on the number of foreign words a given English word may generate, these influences are seen as independent
- Word order is completely ignored (bag of word)
- These slightly unrealistic assumptions simplify the mathematical analysis tremendously: Given a model and a sentence pair (f,e), estimated counts for the events can be obtained in closed form.

IBM Model I

UNIVERSITÄT DES SAARLANDES

Joint Probability of alignment and translation:

$$\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} t(f_j | e_{a_j}). \tag{5}$$

Probability of translation:

$$\Pr(\mathbf{f} | \mathbf{e}) = \frac{\epsilon}{(l+1)^m} \sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} \prod_{j=1}^{m} t(f_j | e_{a_j}). \tag{6}$$

Can be reorganized into:

$$\sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} \prod_{j=1}^{m} t(f_j | e_{a_j}) = \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j | e_i). \tag{15}$$

Counts for word-pair events can now be collected for foreign words, given bag of English words, but independent of foreign context

IBM Model II

- Builds on Model I, but has some limited support for preserving word order
- The probability that a foreign word at position i is generated by an English word at position j is moderated by a alignment probability that depends on i and j, which is learned from the data.
- This has the effect that alignments near the diagonal are preferred over alignments that would require displacement.
- Other problems of Model I remain.

UNIVERSITÄT DES SAARLANDES

IBM Model III

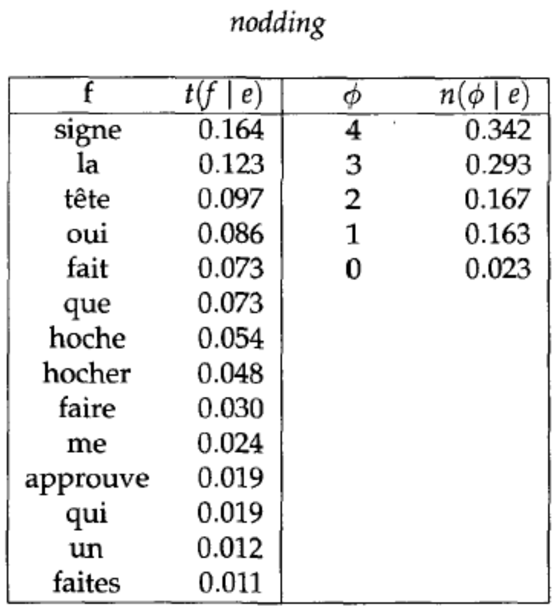
- Introduces the concept of *fertility:* For each of the English words, a number of resulting foreign words is generated (e.g. *not* may give rise to two foreign words *ne..pas,* hence P(fertility=2|not) is high).
- Resulting words are generated independently up to the chosen number; hence *not* may as well generate *ne..ne* or *pas..pas*
- This plausible refinement destroys some of the computational simplicity of Models I and II: Summation over all possible alignments can no more be done efficiently.
- Exact computation for an overly simplified model is therefore replaced by approximation for a better model:
- Instead of using all possible alignments, a smaller set of good alignments (called Viterbi alignments) is generated that are much more likely than the ones that are left out, hence the impact of the error should be small.

IBM Model III

UNIVERSITÄT
DES
SAARLANDES

Examples of translation probabilities/fertilities

nodding → faire (un) signe de tête affirmativ/que oui/…



nodding

f	$t(f \mid e)$	ϕ	$n(\phi \mid e)$
signe	0.164	4	0.342
la	0.123	3	0.293
tête	0.097	2	0.167
oui	0.086	1	0.163
fait	0.073	0	0.023
que	0.073		
hoche	0.054		
hocher	0.048		
faire	0.030		
me	0.024		
approuve	0.019		
qui	0.019		
un	0.012		
faites	0.011		

Figure 6
Translation and fertility probabilities for *nodding*.

IBM Model IV

- Alleviates the problem that up to Model III, movement of a longer phrase as a whole is difficult, as each word is placed (and has to be moved) independently.
- foreign words that originate from the same English source are now placed collectively (starting with a “head”, and placing the remaining words nearby)
- The relation between heads and subordinate words is moderated via a classification of words into 50 classes. This classification is induced from un-annotated text.

IBM Model V

- This solves the technical problem that models III and IV are deficient, i.e. the placement of foreign words is modeled in such a way that the foreign sentence could e.g. have three second words, but no first word.
- Model V makes sure that the resulting sentence is indeed a string. This refinement again increases the computational burden of finding good alignments, so it is only reasonable after some iterations through earlier models have generated strong lexical preferences that constrain the search space.

IBM Models form a chain…

… designed such that data generated in each step can also be used by the next level. Hence the idea of the EM algorithm is generalized to iteration along this cascade

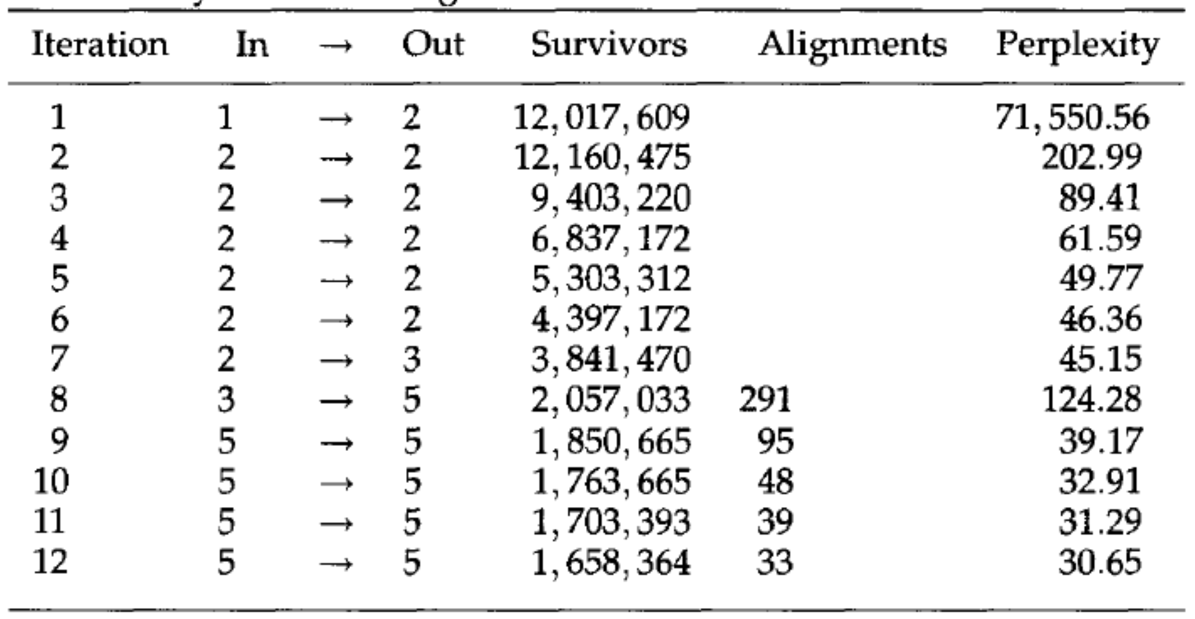


Table 1
A summary of the training iterations.

Iteration	In	→	Out	Survivors	Alignments	Perplexity
1	1	→	2	12,017,609		71,550.56
2	2	→	2	12,160,475		202.99
3	2	→	2	9,403,220		89.41
4	2	→	2	6,837,172		61.59
5	2	→	2	5,303,312		49.77
6	2	→	2	4,397,172		46.36
7	2	→	3	3,841,470		45.15
8	3	→	5	2,057,033	291	124.28
9	5	→	5	1,850,665	95	39.17
10	5	→	5	1,763,665	48	32.91
11	5	→	5	1,703,393	39	31.29
12	5	→	5	1,658,364	33	30.65

This chain is extended…

… in more recent work in a similar spirit. Koehn uses the outcome of GIZA++ (which goes up to model V), but improves and re-interprets this output in several ways:

- Integration of IBM models for both directions into a joint alignment
- Reinforcing consistent parts, removing inconsistent parts
- Extension of the common core to an exhaustive alignment using various heuristics

This chain could be extended…

… in many more ways:

- Given more than n>2 languages, we can compute n*(n-1) directed alignments and combine them to distinguish various levels of confidence
- There is no reason why existing linguistic knowledge for any of the involved languages could not be used to make final alignments still much better
- This would significantly alleviate the severe impact of certain systematic alignment errors in the current SMT approach

Summary of IBM models

IBM translation models

- **Model I:**
  - **Generate foreign words independently, each depending on 0 or 1 English words**
  - **Ignore word order**
- **Model II:**
  - **Position of foreign words depends on position of English origin**
- **Model III:**
  - **English words have "fertilities" to determine number of foreign words they generate**
- **Model IV:**
  - **Groups of words are moved to their target location as a whole**
- **Model V:**
  - **Avoid loss of probabilities on impossible strings**

Models can "feed each other": Alignments of Model k can be used to estimate parameters of Model k+1

To use these models on real data, the best place to start are the baseline systems for the shared tasks listed on http://www.statmt.org/

References

UNIVERSITÄT DES SAARLANDES

Sentence alignment

- ❑ Manning, C. & Schütze, H. (1999): *Foundations of Statistical Natural Language Processing*. MIT Press, Chap. 13.1 und 13.2.
- ❑ Gale, W.A. & Church, K.W. (1993): A program for aligning sentences in bilingual corpora. *Computational Linguistics* 19,p. 75-102.
- ❑ Kay, M. & Röscheisen, M. (1993): Text-translation alignment. *Computational Linguistics* 19, p. 121-142.

References

UNIVERSITÄT DES SAARLANDES

Word alignment

- Brown, Cocke, Della Pietra, Della Pietra, Jelinek, Lafferty, Mercer, and Roossin (1993): A Statistical Approach to Machine Translation. In: Computational Linguistics 16,2.
- Franz Josef Och, Christoph Tillmann, and Hermann Ney (1999):Improved alignment models for statistical machine translation. In: Proceedings of EMNLP
- Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase based translation. In: Proceedings of HLT-NAACL.
- Philipp Koehn (2004). Pharaoh Training Manual. Unpublished manuscript. Available from http://www.statmt.org/wmt06/shared-task/training-release-1.3.tgz

Exercises

Exercise 1:

- Implement the LCS procedure in a programming language of your choice
- Try it on at least 20 pairs of strings, make sure that the boundary conditions work

Exercise 2:

- Generalize the result of Exercise 1 to use a different error metric, where a replacement of a vowel by another vowel or of a consonant by another consonant count only as 0.5 errors. Which of the test cases from Exercise 1 are affected?

Send the code and the results via email to eisele@dfki.de