Statistical Machine Translation LING-462/COSC-482 Week 3: Language Models and Word-based models

Achim Ruopp

achim.ruopp@Georgetown.edu

Agenda

- Language in 10 minutes
  - Harry Eldridge: Lojban
- Word Alignment

- Break -

- Language Models
- Computing Environment
- Homework 2 Assignment

Lexical Translation

- How to translate a word $\rightarrow$ 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: In all lectures, we 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

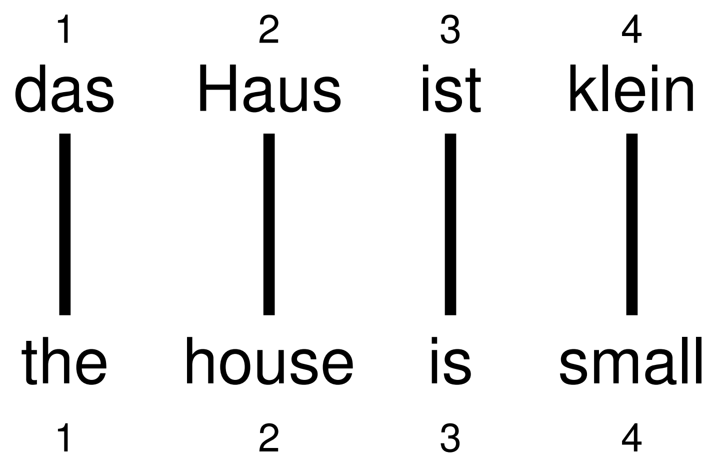
Estimate Translation Probabilities

Maximum likelihood estimation

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

Alignment

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



- Word positions are numbered 1–4

Alignment Function

- Formalizing alignment with an alignment function

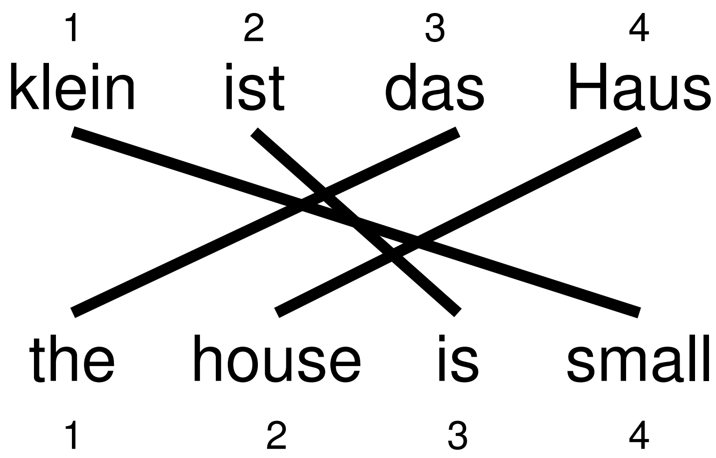
- Mapping an English target word at position i to a German source word at position j with a function $a : i \rightarrow j$

- Example

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

Reordering

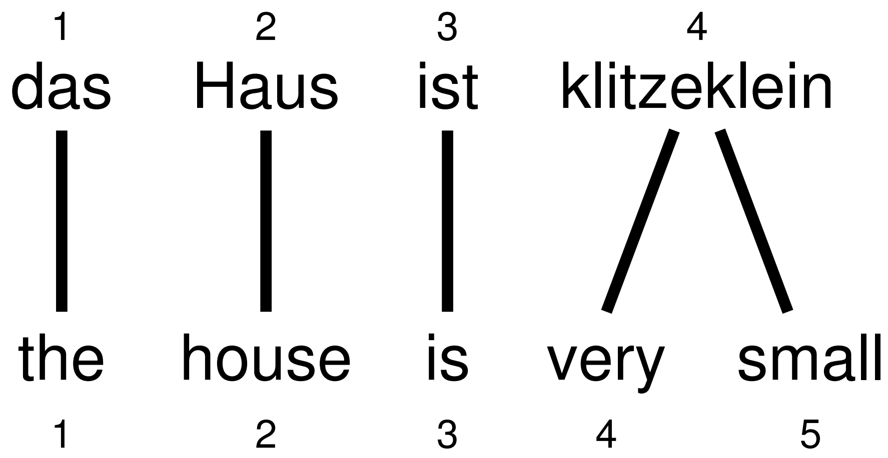
Words may be reordered during translation



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

One-to-Many Translation

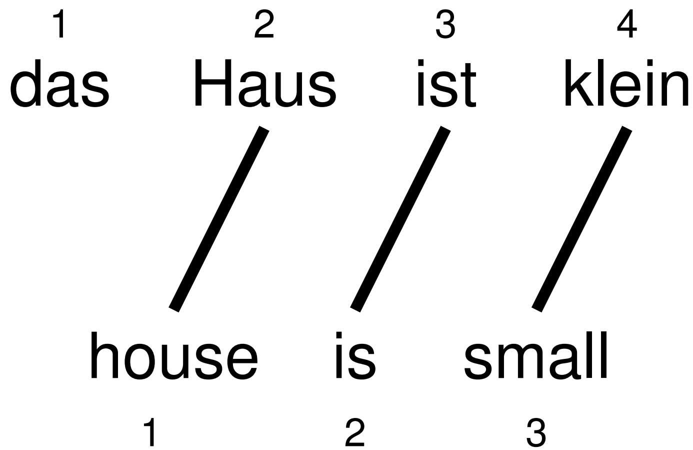
A source word may translate into multiple target words



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

Dropping Words

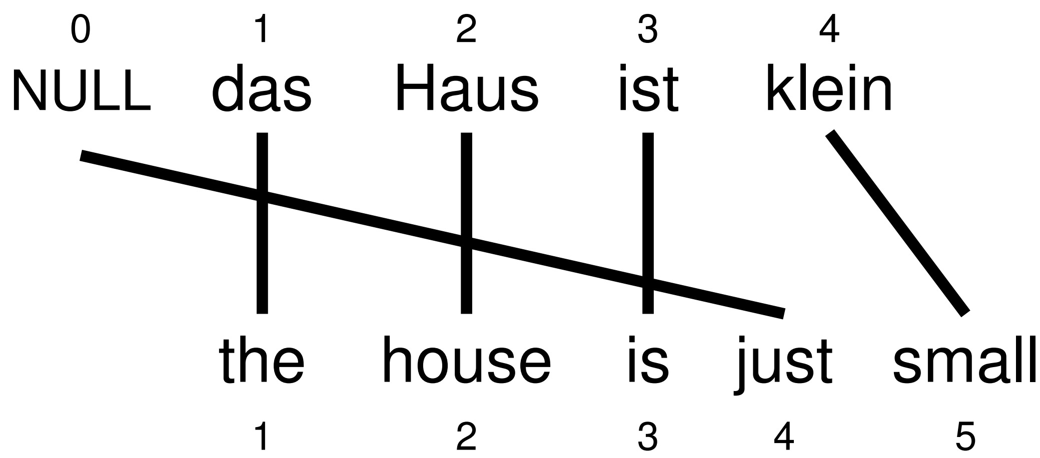
Words may be dropped when translated
(German article das is dropped)



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

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

$$\begin{aligned} p(e,a|f) &= \frac{\epsilon}{4^3} \times t(\text{the}|\text{das}) \times t(\text{house}|\text{Haus}) \times t(\text{is}|\text{ist}) \times t(\text{small}|\text{klein}) \\ &= \frac{\epsilon}{4^3} \times 0.7 \times 0.8 \times 0.8 \times 0.4 \\ &= 0.0028\epsilon \end{aligned}$$

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

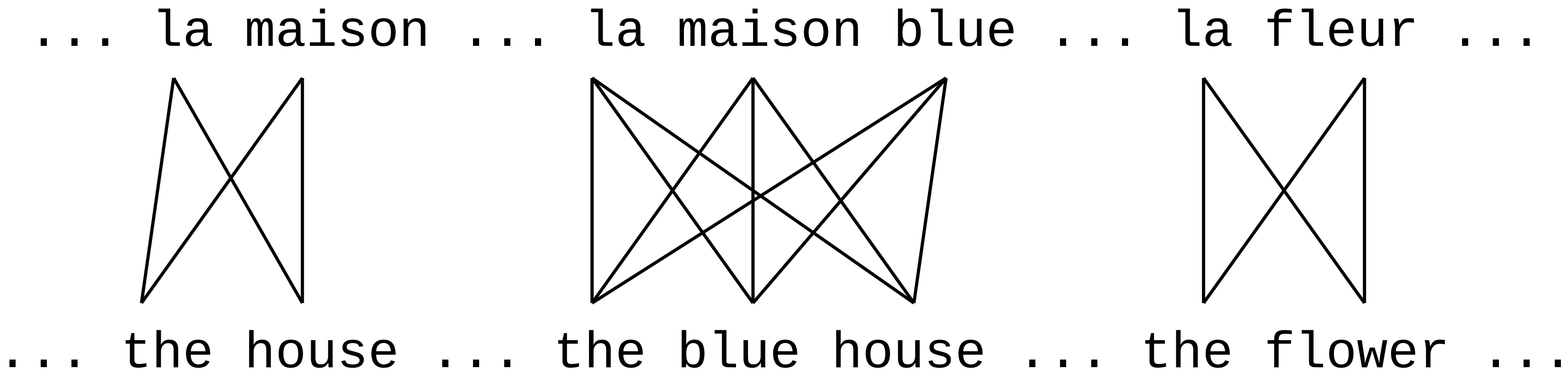
- Chicken and egg problem
  - if we had the *alignments*,
    $\rightarrow$ we could estimate the *parameters* of our generative model
  - if we had the *parameters*,
    $\rightarrow$ we could estimate the *alignments*

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
  1. initialize model parameters (e.g. uniform)
  2. assign probabilities to the missing data
  3. estimate model parameters from completed data
  4. iterate steps 2–3 until convergence

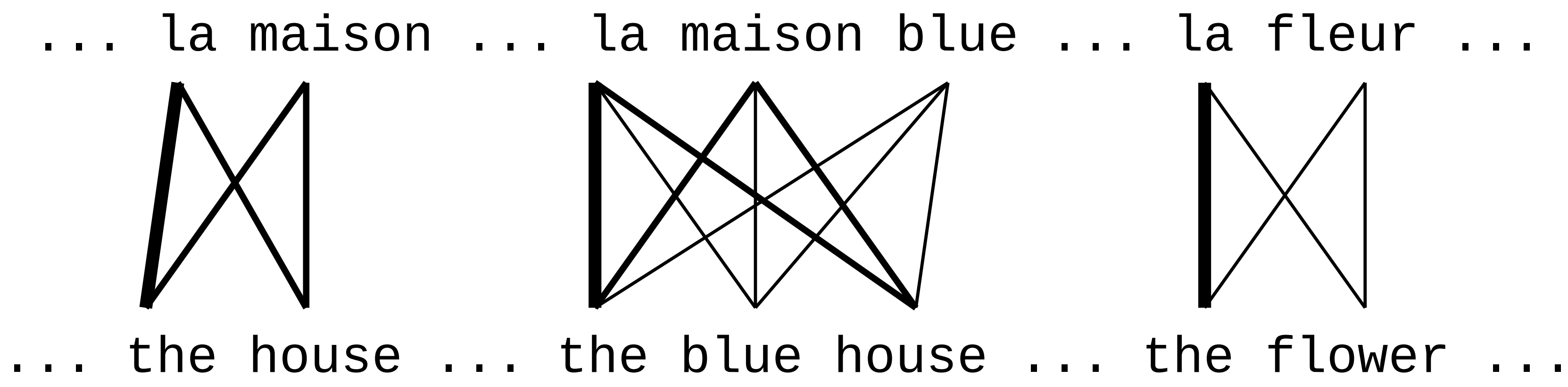
EM Algorithm



- Initial step: all alignments equally likely

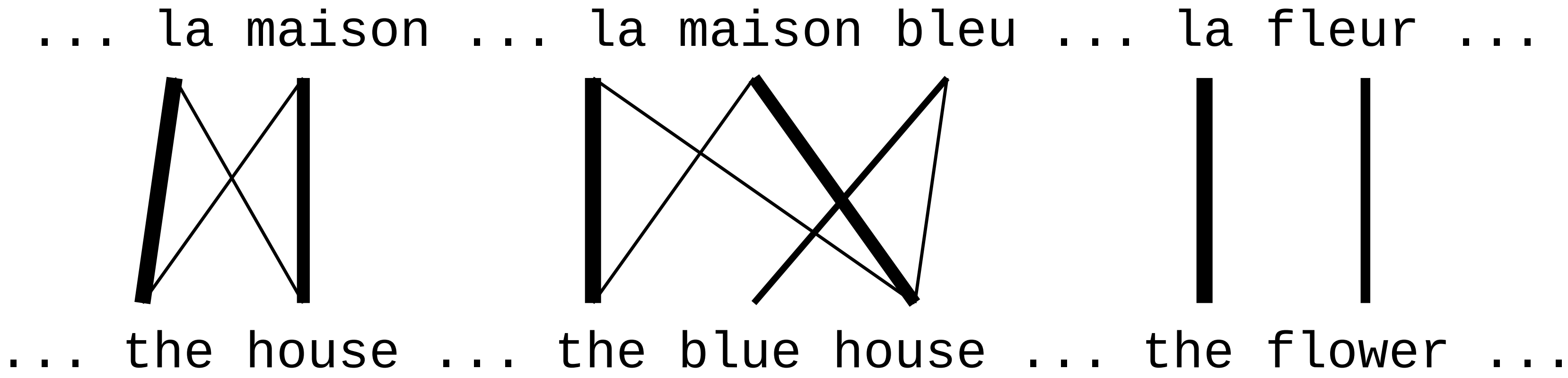
- Model learns that, e.g., la is often aligned with the

EM Algorithm



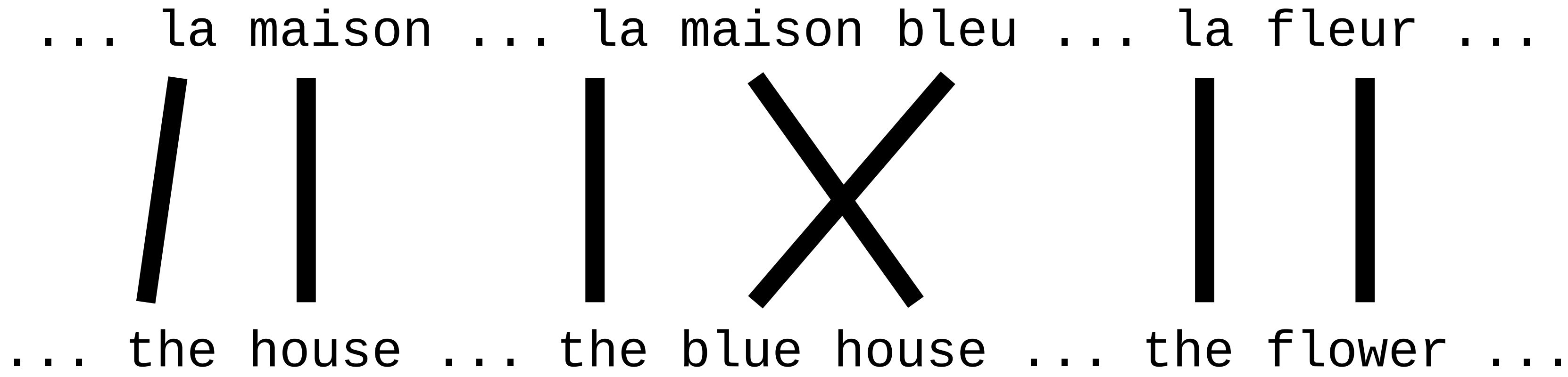
- After one iteration
- Alignments, e.g., between la and the are more likely

EM Algorithm



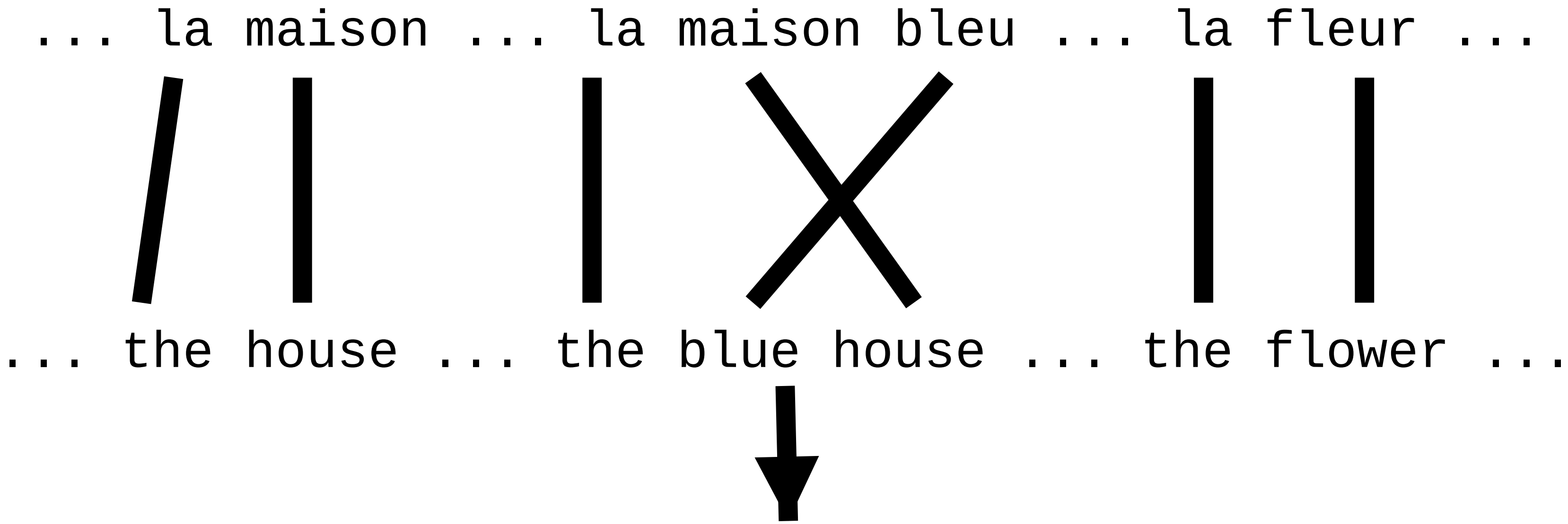
- After another iteration
- It becomes apparent that alignments, e.g., between fleur and flower are more likely (pigeon hole principle)

EM Algorithm



- Convergence
- Inherent hidden structure revealed by EM

EM Algorithm




```
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

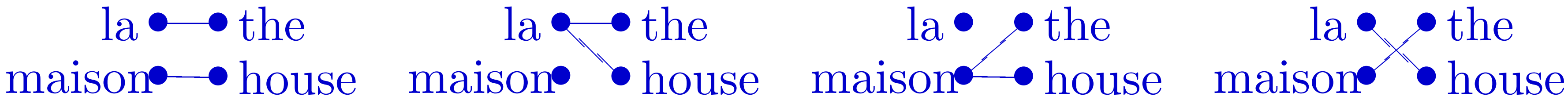
- We need to be able to compute:
  - Expectation-Step: probability of alignments
  - Maximization-Step: count collection

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

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

IBM Model 1 and EM: Expectation Step

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

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

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

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

$\rightarrow$ this makes IBM Model 1 estimation tractable

The Trick

(case $l_e = l_f = 2$)

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

$$= t(e_1|f_0)\ t(e_2|f_0) + t(e_1|f_0)\ t(e_2|f_1) + t(e_1|f_0)\ t(e_2|f_2) +$$

$$+ t(e_1|f_1)\ t(e_2|f_0) + t(e_1|f_1)\ t(e_2|f_1) + t(e_1|f_1)\ t(e_2|f_2) +$$

$$+ t(e_1|f_2)\ t(e_2|f_0) + t(e_1|f_2)\ t(e_2|f_1) + t(e_1|f_2)\ t(e_2|f_2) =$$

$$= t(e_1|f_0)\ (t(e_2|f_0) + t(e_2|f_1) + t(e_2|f_2)) +$$

$$+ t(e_1|f_1)\ (t(e_2|f_1) + t(e_2|f_1) + t(e_2|f_2)) +$$

$$+ t(e_1|f_2)\ (t(e_2|f_2) + t(e_2|f_1) + t(e_2|f_2)) =$$

$$= (t(e_1|f_0) + t(e_1|f_1) + t(e_1|f_2))\ (t(e_2|f_2) + t(e_2|f_1) + t(e_2|f_2))$$

IBM Model 1 and EM: Expectation Step

- Combine what we have:

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

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:

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

IBM Model 1 and EM: Maximization Step

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

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

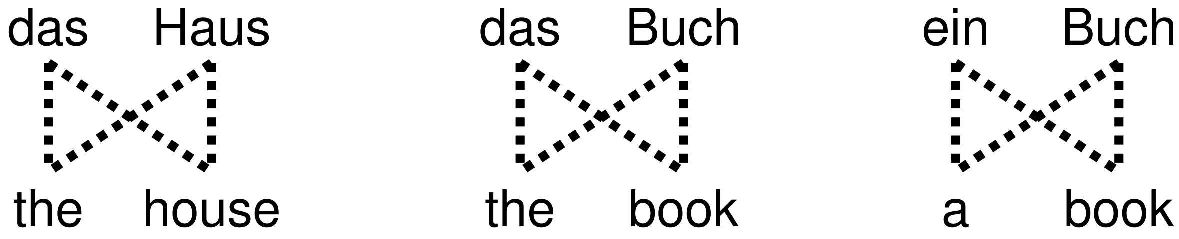
IBM Model 1 and EM: Pseudocode

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

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

```
initialize t(e|f) uniformly
while not converged do
    // initialize
    count(e|f) = 0 for all e, f
    total(f) = 0 for all f
    for all sentence pairs (e,f) do
        // compute normalization
        for all words e in e do
            s-total(e) = 0
            for all words f in f do
                s-total(e) += t(e|f)
            end for
        end for
        // collect counts
        for all words e in e do
            for all words f in f do
                count(e|f) += t(e|f) / s-total(e)
                total(f) += t(e|f) / s-total(e)
            end for
        end for
    end for
    // estimate probabilities
    for all foreign words f do
        for all English words e do
            t(e|f) = count(e|f) / total(f)
        end for
    end for
end while
```

Convergence



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

Perplexity

- How well does the model fit the data?
- Perplexity: derived from probability of the training data according to the model

$$\log_2 PP = -\sum_s \log_2 p(\mathbf{e}_s|\mathbf{f}_s)$$

- Example (ϵ=1)

	initial	1st it.	2nd it.	3rd it.	...	final
$p(\text{the haus}|\text{das haus})$	0.0625	0.1875	0.1905	0.1913	...	0.1875
$p(\text{the book}|\text{das buch})$	0.0625	0.1406	0.1790	0.2075	...	0.25
$p(\text{a book}|\text{ein buch})$	0.0625	0.1875	0.1907	0.1913	...	0.1875
perplexity	4095	202.3	153.6	131.6	...	113.8

Ensuring Fluent Output

- Our translation model cannot decide between small and little
- Sometime one is preferred over the other:
  - small step: 2,070,000 occurrences in the Google index
  - little step: 257,000 occurrences in the Google index
- Language model
  - estimate how likely a string is English
  - based on n-gram statistics

$$\begin{aligned} p(\mathbf{e}) &= p(e_1, e_2, ..., e_n) \\ &= p(e_1)p(e_2|e_1)...p(e_n|e_1, e_2, ..., e_{n-1}) \\ &\simeq p(e_1)p(e_2|e_1)...p(e_n|e_{n-2}, e_{n-1}) \end{aligned}$$

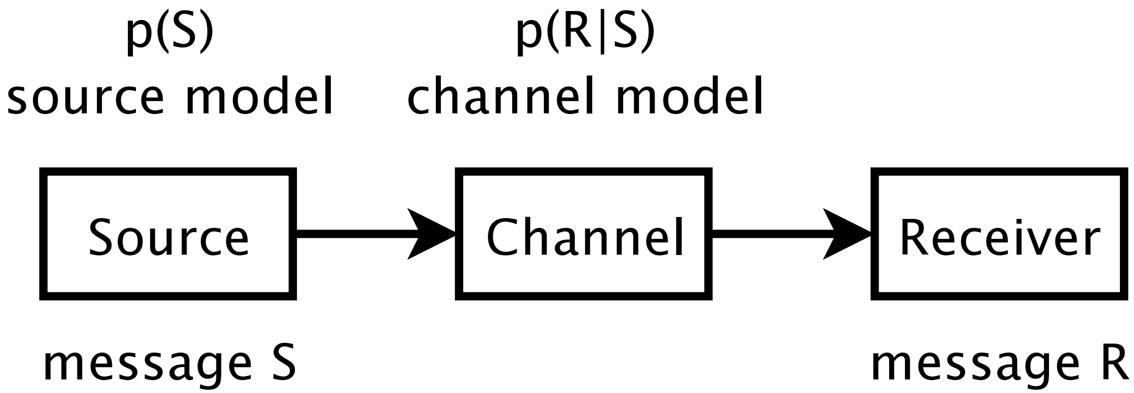
Noisy Channel Model

- We would like to integrate a language model
- Bayes rule

$$\text{argmax}_{\mathbf{e}}\, p(\mathbf{e}|\mathbf{f}) = \text{argmax}_{\mathbf{e}} \frac{p(\mathbf{f}|\mathbf{e})\, p(\mathbf{e})}{p(\mathbf{f})}$$

$$= \text{argmax}_{\mathbf{e}}\, p(\mathbf{f}|\mathbf{e})\, p(\mathbf{e})$$

Noisy Channel Model



- Applying Bayes rule also called noisy channel model
  - we observe a distorted message R (here: a foreign string **f**)
  - we have a model on how the message is distorted (here: translation model)
  - we have a model on what messages are probably (here: language model)
  - we want to recover the original message S (here: an English string **e**)

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

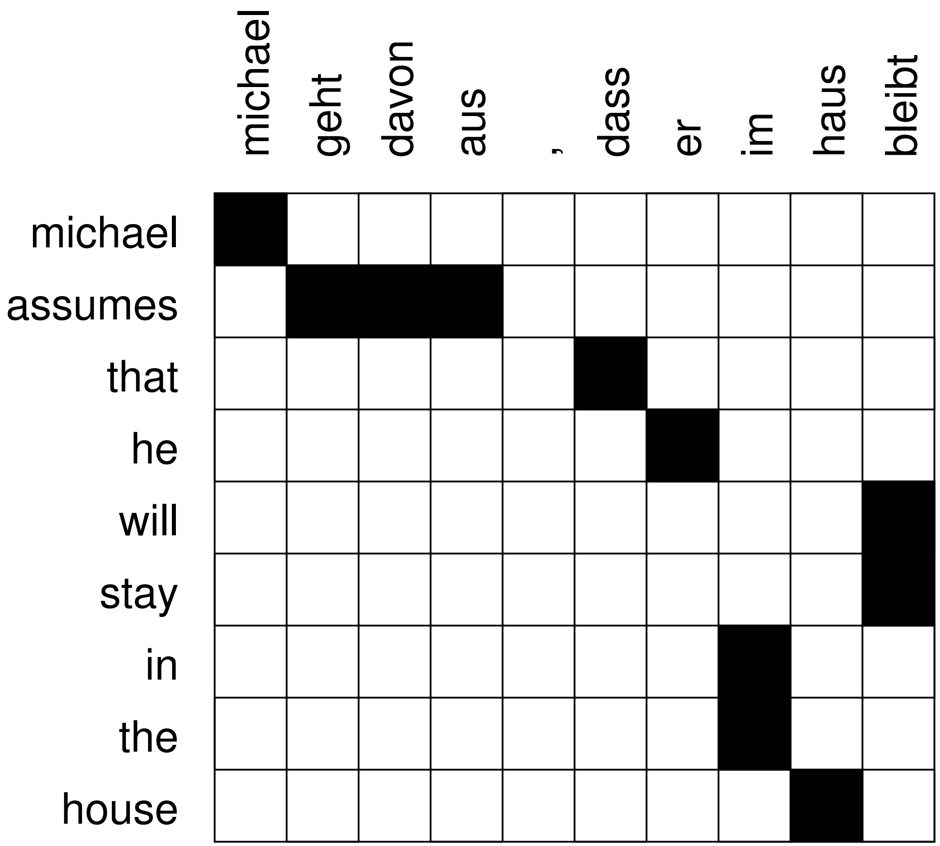
- Compuationally biggest change in Model 3
  - trick to simplify estimation does not work anymore
  - $\rightarrow$ exhaustive count collection becomes computationally too expensive
  - sampling over high probability alignments is used instead

Conclusion

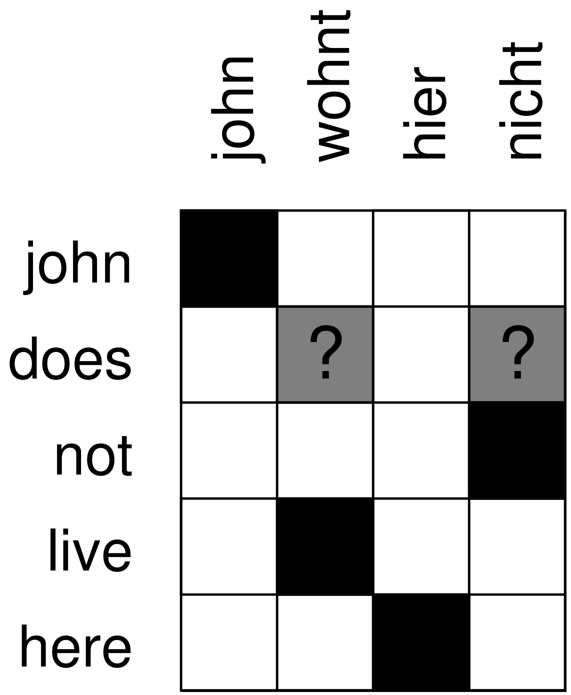
- IBM Models were the pioneering models in statistical machine translation
- Introduced important concepts
  - generative model
  - EM training
  - reordering models
- Only used for niche applications as translation model
- ... but still in common use for word alignment (e.g., GIZA++ toolkit)

Word Alignment

Given a sentence pair, which words correspond to each other?

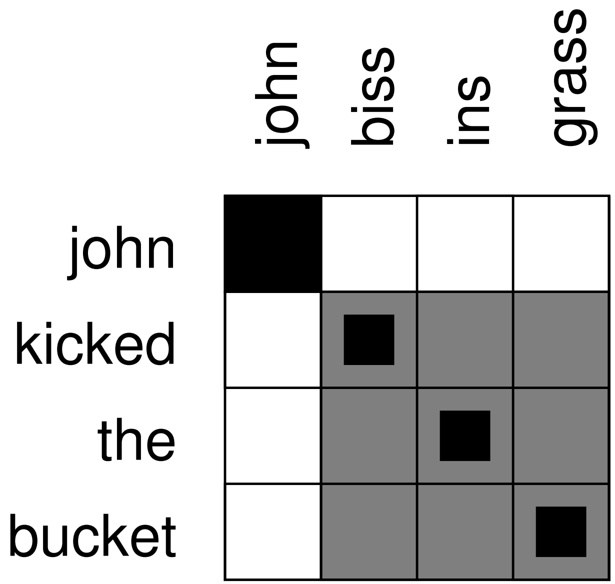


Word Alignment?



Is the English word does aligned to
the German wohnt (verb) or nicht (negation) or neither?

Word Alignment?



How do the idioms kicked the bucket and biss ins grass match up?
Outside this exceptional context, bucket is never a good translation for grass

Measuring Word Alignment Quality

- Manually align corpus with *sure* (S) and *possible* (P) alignment points ($S \subseteq P$)
- Common metric for evaluation word alignments: Alignment Error Rate (AER)

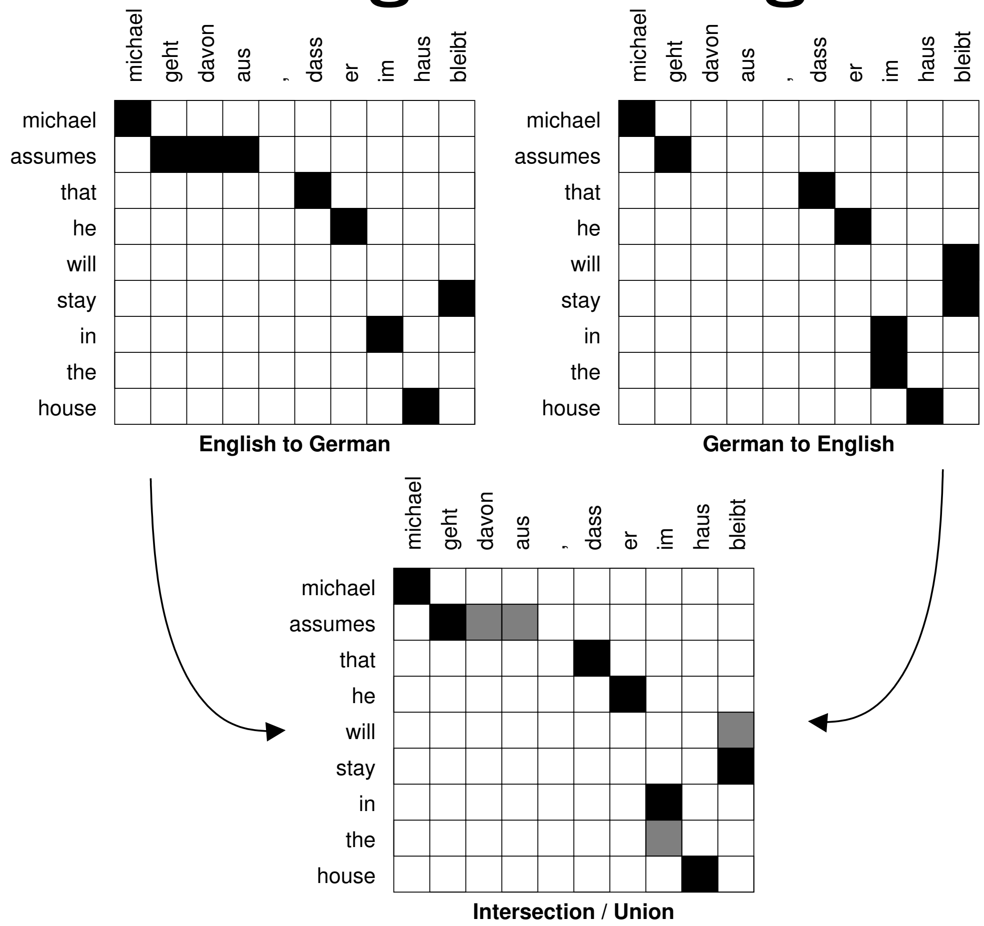
$$\text{AER}(S, P; A) = \frac{|A \cap S| + |A \cap P|}{|A| + |S|}$$

- AER $= 0$: alignment A matches all sure, any possible alignment points
- However: different applications require different precision/recall trade-offs

Word Alignment with IBM Models

- IBM Models create a **many-to-one** mapping
  - words are aligned using an alignment function
  - a function may return the same value for different input (one-to-many mapping)
  - a function can not return multiple values for one input (no many-to-one mapping)
- Real word alignments have **many-to-many** mappings

Symmetrizing Word Alignments



- Intersection of GIZA++ bidirectional alignments
- Grow additional alignment points [Och and Ney, CompLing2003]

Language models

- **Language models** answer the question:

  How likely is a string of English words good English?

- Help with reordering

$$p_{\text{LM}}(\text{the house is small}) > p_{\text{LM}}(\text{small the is house})$$

- Help with word choice

$$p_{\text{LM}}(\text{I am going home}) > p_{\text{LM}}(\text{I am going house})$$

N-Gram Language Models

- Given: a string of English words $W = w_1, w_2, w_3, ..., w_n$

- Question: what is $p(W)$?

- Sparse data: Many good English sentences will not have been seen before

$\rightarrow$ Decomposing $p(W)$ using the chain rule:

$$p(w_1, w_2, w_3, ..., w_n) = p(w_1)\ p(w_2|w_1)\ p(w_3|w_1, w_2)...p(w_n|w_1, w_2, ...w_{n-1})$$

(not much gained yet, $p(w_n|w_1, w_2, ...w_{n-1})$ is equally sparse)

Markov Chain

- **Markov assumption**:
  - only previous history matters
  - limited memory: only last k words are included in history (older words less relevant)

  $\rightarrow$ k**th order Markov model**

- For instance 2-gram language model:

$$p(w_1, w_2, w_3, ..., w_n) \simeq p(w_1)\ p(w_2|w_1)\ p(w_3|w_2) ... p(w_n|w_{n-1})$$

- What is conditioned on, here w_{i-1} is called the **history**

Estimating N-Gram Probabilities

- Maximum likelihood estimation

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

- Collect counts over a large text corpus

- Millions to billions of words are easy to get
  (trillions of English words available on the web)

Example: 3-Gram

- Counts for trigrams and estimated word probabilities

the green (total: 1748)

word	c.	prob.
paper	801	0.458
group	640	0.367
light	110	0.063
party	27	0.015
ecu	21	0.012

the red (total: 225)

word	c.	prob.
cross	123	0.547
tape	31	0.138
army	9	0.040
card	7	0.031
,	5	0.022

the blue (total: 54)

word	c.	prob.
box	16	0.296
.	6	0.111
flag	6	0.111
,	3	0.056
angel	3	0.056

- 225 trigrams in the Europarl corpus start with the red
- 123 of them end with cross

$\rightarrow$ maximum likelihood probability is $\frac{123}{225} = 0.547$.

How good is the LM?

- A good model assigns a text of real English W a high probability
- This can be also measured with cross entropy:

$$H(W) = \frac{1}{n} \log p(W_1^n)$$

- Or, **perplexity**

$$\text{perplexity}(W) = 2^{H(W)}$$

Example: 4-Gram

prediction	p_{LM}	$-\log_2 p_{\text{LM}}$
$p_{\text{LM}}(\text{i}\vert\text{</s><s>})$	0.109	3.197
$p_{\text{LM}}(\text{would}\vert\text{<s>i})$	0.144	2.791
$p_{\text{LM}}(\text{like}\vert\text{i would})$	0.489	1.031
$p_{\text{LM}}(\text{to}\vert\text{would like})$	0.905	0.144
$p_{\text{LM}}(\text{commend}\vert\text{like to})$	0.002	8.794
$p_{\text{LM}}(\text{the}\vert\text{to commend})$	0.472	1.084
$p_{\text{LM}}(\text{rapporteur}\vert\text{commend the})$	0.147	2.763
$p_{\text{LM}}(\text{on}\vert\text{the rapporteur})$	0.056	4.150
$p_{\text{LM}}(\text{his}\vert\text{rapporteur on})$	0.194	2.367
$p_{\text{LM}}(\text{work}\vert\text{on his})$	0.089	3.498
$p_{\text{LM}}(.\vert\text{his work})$	0.290	1.785
$p_{\text{LM}}(\text{</s>}\vert\text{work .})$	0.99999	0.000014
	average	2.634

Comparison 1–4-Gram

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

Unseen N-Grams

- We have seen i like to in our corpus

- We have never seen i like to smooth in our corpus

$\rightarrow p(\text{smooth}|\text{i like to}) = 0$

- Any sentence that includes i like to smooth will be assigned probability 0

Add-One Smoothing

- For all possible n-grams, add the count of one.

$$p = \frac{c+1}{n+v}$$

  - c = count of n-gram in corpus
  - n = count of history
  - v = vocabulary size

- But there are many more unseen n-grams than seen n-grams

- Example: Europarl 2-bigrams:
  - $86,700$ distinct words
  - $86,700^2 = 7,516,890,000$ possible bigrams
  - but only about $30,000,000$ words (and bigrams) in corpus

Add-α Smoothing

- Add $\alpha < 1$ to each count

$$p = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?

- Could be optimized on held-out set

Example: 2-Grams in Europarl

Count	Adjusted count		Test count
c	$(c+1)\frac{n}{n+v^2}$	$(c+\alpha)\frac{n}{n+\alpha v^2}$	t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

- Add-α smoothing with $\alpha = 0.00017$
- t_c are average counts of n-grams in test set that occurred c times in corpus

Deleted Estimation

- Estimate true counts in held-out data
  - split corpus in two halves: training and held-out
  - counts in training $C_t(w_1, ..., w_n)$
  - number of ngrams with training count r: N_r
  - total times ngrams of training count r seen in held-out data: T_r

- Held-out estimator:

$$p_h(w_1, ..., w_n) = \frac{T_r}{N_r N} \quad \text{where } count(w_1, ..., w_n) = r$$

- Both halves can be switched and results combined

$$p_h(w_1, ..., w_n) = \frac{T_r^1 + T_r^2}{N(N_r^1 + N_r^2)} \quad \text{where } count(w_1, ..., w_n) = r$$

Good-Turing Smoothing

- Adjust actual counts r to expected counts r^* with formula

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

  - N_r number of n-grams that occur exactly r times in corpus
  - N_0 total number of n-grams

Good-Turing for 2-Grams in Europarl

Count	Count of counts	Adjusted count	Test count
r	N_r	r^*	t
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

adjusted count fairly accurate when compared against the test count

Derivation of Good-Turing

- A specific n-gram α occurs with (unknown) probability p in the corpus
- Assumption: all occurrences of an n-gram α are independent of each other
- Number of times α occurs in corpus follows binomial distribution

$$p(c(\alpha) = r) = b(r; N, p_i) = \binom{N}{r} p^r (1-p)^{N-r}$$

Derivation of Good-Turing (2)

- Goal of Good-Turing smoothing: compute *expected count* c^*
- Expected count can be computed with help from binomial distribution:

$$
\begin{aligned}
E(c^*(\alpha)) &= \sum_{r=0}^{N} r \; p(c(\alpha) = r) \\
&= \sum_{r=0}^{N} r \binom{N}{r} p^r (1-p)^{N-r}
\end{aligned}
$$

- Note again: p is unknown, we cannot actually compute this

Derivation of Good-Turing (3)

- Definition: expected number of n-grams that occur r times: $E_N(N_r)$

- We have s different n-grams in corpus
  - let us call them $\alpha_1, ..., \alpha_s$
  - each occurs with probability $p_1, ..., p_s$, respectively

- Given the previous formulae, we can compute

$$\begin{aligned} E_N(N_r) &= \sum_{i=1}^{s} p(c(\alpha_i) = r) \\ &= \sum_{i=1}^{s} \binom{N}{r} p_i^r (1 - p_i)^{N-r} \end{aligned}$$

- Note again: p_i is unknown, we cannot actually compute this

Derivation of Good-Turing (4)

- Reflection
  - we derived a formula to compute $E_N(N_r)$
  - we have N_r
  - for small r: $E_N(N_r) \simeq N_r$

- Ultimate goal compute expected counts c^*, given actual counts c

$$E(c^*(\alpha)|c(\alpha) = r)$$

Derivation of Good-Turing (5)

- For a particular n-gram α, we know its actual count r
- Any of the n-grams α_i may occur r times
- Probability that α is one specific α_i

$$p(\alpha = \alpha_i | c(\alpha) = r) = \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- Expected count of this n-gram α

$$E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N\, p_i\, p(\alpha = \alpha_i | c(\alpha) = r)$$

Derivation of Good-Turing (6)

- Combining the last two equations:

$$E(c^*(\alpha)|c(\alpha)=r) = \sum_{i=1}^{s} N\, p_i \frac{p(c(\alpha_i)=r)}{\sum_{j=1}^{s} p(c(\alpha_j)=r)}$$
$$= \frac{\sum_{i=1}^{s} N\, p_i\, p(c(\alpha_i)=r)}{\sum_{j=1}^{s} p(c(\alpha_j)=r)}$$

- We will now transform this equation to derive Good-Turing smoothing

Derivation of Good-Turing (7)

- Repeat:

$$E(c^*(\alpha)|c(\alpha) = r) \; = \frac{\sum_{i=1}^{s} N \; p_i \; p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- Denominator is our definition of expected counts $E_N(N_r)$

Derivation of Good-Turing (8)

- Numerator:

$$
\begin{aligned}
\sum_{i=1}^{s} N\, p_i\, p(c(\alpha_i) = r) &= \sum_{i=1}^{s} N\, p_i \binom{N}{r} p_i^r (1-p_i)^{N-r} \\
&= N\, \frac{N!}{N-r!r!} p_i^{r+1} (1-p_i)^{N-r} \\
&= N\, \frac{(r+1)}{N+1} \frac{N+1!}{N-r!r+1!} p_i^{r+1} (1-p_i)^{N-r} \\
&= (r+1)\, \frac{N}{N+1} E_{N+1}(N_{r+1}) \\
&\simeq (r+1)\, E_{N+1}(N_{r+1})
\end{aligned}
$$

Derivation of Good-Turing (9)

- Using the simplifications of numerator and denominator:

$$\begin{aligned} r^* &= E(c^*(\alpha)|c(\alpha) = r) \\ &= \frac{(r+1)\ E_{N+1}(N_{r+1})}{E_N(N_r)} \\ &\simeq (r+1)\frac{N_{r+1}}{N_r} \end{aligned}$$

- QED

Back-Off

- In given corpus, we may never observe
  - Scottish beer drinkers
  - Scottish beer eaters

- Both have count 0

  $\rightarrow$ our smoothing methods will assign them same probability

- Better: backoff to bigrams:
  - beer drinkers
  - beer eaters

Interpolation

- Higher and lower order n-gram models have different strengths and weaknesses
  - high-order n-grams are sensitive to more context, but have sparse counts
  - low-order n-grams consider only very limited context, but have robust counts

- Combine them

$$
\begin{aligned}
p_I(w_3|w_1, w_2) = \quad & \lambda_1 \, p_1(w_3) \\
& \times \lambda_2 \, p_2(w_3|w_2) \\
& \times \lambda_3 \, p_3(w_3|w_1, w_2)
\end{aligned}
$$

Recursive Interpolation

- We can trust some histories $w_{i-n+1}, ..., w_{i-1}$ more than others
- Condition interpolation weights on history: $\lambda_{w_{i-n+1},...,w_{i-1}}$
- Recursive definition of interpolation

$$
\begin{aligned}
p_n^I(w_i|w_{i-n+1}, ..., w_{i-1}) = {} & \lambda_{w_{i-n+1},...,w_{i-1}}\ p_n(w_i|w_{i-n+1}, ..., w_{i-1}) + \\
& + \ (1 - \lambda_{w_{i-n+1},...,w_{i-1}})\ p_{n-1}^I(w_i|w_{i-n+2}, ..., w_{i-1})
\end{aligned}
$$

Back-Off

- Trust the highest order language model that contains n-gram

$$p_n^{BO}(w_i|w_{i-n+1},...,w_{i-1}) =$$

$$= \begin{cases} \alpha_n(w_i|w_{i-n+1},...,w_{i-1}) \\ \qquad\qquad \text{if count}_n(w_{i-n+1},...,w_i) > 0 \\ d_n(w_{i-n+1},...,w_{i-1})\ p_{n-1}^{BO}(w_i|w_{i-n+2},...,w_{i-1}) \\ \qquad\qquad \text{else} \end{cases}$$

- Requires
  - adjusted prediction model $\alpha_n(w_i|w_{i-n+1},...,w_{i-1})$
  - discounting function $d_n(w_1,...,w_{n-1})$

Back-Off with Good-Turing Smoothing

- Previously, we computed n-gram probabilities based on relative frequency

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

- Good Turing smoothing adjusts counts c to expected counts c^*

$$\text{count}^*(w_1, w_2) \leq \text{count}(w_1, w_2)$$

- We use these expected counts for the prediction model (but 0^* remains 0)

$$\alpha(w_2|w_1) = \frac{\text{count}^*(w_1, w_2)}{\text{count}(w_1)}$$

- This leaves probability mass for the discounting function

$$d_2(w_1) = 1 - \sum_{w_2} \alpha(w_2|w_1)$$

Diversity of Predicted Words

- Consider the bigram histories spite and constant
  - both occur 993 times in Europarl corpus
  - only 9 different words follow spite
    almost always followed by of (979 times), due to expression in spite of
  - 415 different words follow constant
    most frequent: and (42 times), concern (27 times), pressure (26 times),
    but huge tail of singletons: 268 different words

- More likely to see new bigram that starts with constant than spite

- Witten-Bell smoothing considers diversity of predicted words

Witten-Bell Smoothing

- Recursive interpolation method
- Number of possible extensions of a history $w_1, ..., w_{n-1}$ in training data

$$N_{1+}(w_1, ..., w_{n-1}, \bullet) = |\{w_n : c(w_1, ..., w_{n-1}, w_n) > 0\}|$$

- Lambda parameters

$$1 - \lambda_{w_1,...,w_{n-1}} = \frac{N_{1+}(w_1, ..., w_{n-1}, \bullet)}{N_{1+}(w_1, ..., w_{n-1}, \bullet) + \sum_{w_n} c(w_1, ..., w_{n-1}, w_n)}$$

Witten-Bell Smoothing: Examples

Let us apply this to our two examples:

$$\begin{aligned}
1 - \lambda_{\textit{spite}} &= \frac{N_{1+}(\text{spite}, \bullet)}{N_{1+}(\text{spite}, \bullet) + \sum_{w_n} c(\text{spite}, w_n)} \\
&= \frac{9}{9 + 993} = 0.00898
\end{aligned}$$

$$\begin{aligned}
1 - \lambda_{\textit{constant}} &= \frac{N_{1+}(\text{constant}, \bullet)}{N_{1+}(\text{constant}, \bullet) + \sum_{w_n} c(\text{constant}, w_n)} \\
&= \frac{415}{415 + 993} = 0.29474
\end{aligned}$$

Diversity of Histories

- Consider the word York
  - fairly frequent word in Europarl corpus, occurs 477 times
  - as frequent as foods, indicates and providers

  $\rightarrow$ in unigram language model: a respectable probability

- However, it almost always directly follows New (473 times)

- Recall: unigram model only used, if the bigram model inconclusive
  - York unlikely second word in unseen bigram
  - in back-off unigram model, York should have low probability

Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account

- Count of histories for a word

$$N_{1+}(\bullet w) \ = |\{w_i : c(w_i, w) > 0\}|$$

- Recall: maximum likelihood estimation of unigram language model

$$p_{ML}(w) = \frac{c(w)}{\sum_i c(w_i)}$$

- In Kneser-Ney smoothing, replace raw counts with count of histories

$$p_{KN}(w) = \frac{N_{1+}(\bullet w)}{\sum_{w_i} N_{1+}(\bullet w_i)}$$

Modified Kneser-Ney Smoothing

- Based on interpolation

$$p_n^{BO}(w_i|w_{i-n+1},...,w_{i-1}) =$$

$$= \begin{cases} \alpha_n(w_i|w_{i-n+1},...,w_{i-1}) \\ \qquad\qquad \text{if count}_n(w_{i-n+1},...,w_i) > 0 \\ d_n(w_{i-n+1},...,w_{i-1})\ p_{n-1}^{BO}(w_i|w_{i-n+2},...,w_{i-1}) \\ \qquad\qquad \text{else} \end{cases}$$

- Requires
  - adjusted prediction model $\alpha_n(w_i|w_{i-n+1},...,w_{i-1})$
  - discounting function $d_n(w_1,...,w_{n-1})$

Formula for α for Highest Order N-Gram Model

- Absolute discounting: subtract a fixed D from all non-zero counts

$$\alpha(w_n|w_1,...,w_{n-1}) = \frac{c(w_1,...,w_n) - D}{\sum_w c(w_1,...,w_{n-1},w)}$$

- Refinement: three different discount values

$$D(c) = \begin{cases} D_1 & \text{if } c = 1 \\ D_2 & \text{if } c = 2 \\ D_{3+} & \text{if } c \geq 3 \end{cases}$$

Discount Parameters

- Optimal discounting parameters D_1, D_2, D_{3+} can be computed quite easily

$$
\begin{aligned}
Y &= \frac{N_1}{N_1 + 2N_2} \\
D_1 &= 1 - 2Y\frac{N_2}{N_1} \\
D_2 &= 2 - 3Y\frac{N_3}{N_2} \\
D_{3+} &= 3 - 4Y\frac{N_4}{N_3}
\end{aligned}
$$

- Values N_c are the counts of n-grams with exactly count c

Formula for d for Highest Order N-Gram Model

- Probability mass set aside from seen events

$$d(w_1, ..., w_{n-1}) = \frac{\sum_{i \in \{1,2,3+\}} D_i N_i(w_1, ..., w_{n-1} \bullet)}{\sum_{w_n} c(w_1, ..., w_n)}$$

- N_i for $i \in \{1, 2, 3+\}$ are computed based on the count of extensions of a history $w_1, ..., w_{n-1}$ with count 1, 2, and 3 or more, respectively.

- Similar to Witten-Bell smoothing

Formula for α for Lower Order N-Gram Models

- Recall: base on count of histories $N_{1+}(\bullet w)$ in which word may appear, not raw counts.

$$\alpha(w_n|w_1, ..., w_{n-1}) = \frac{N_{1+}(\bullet w_1, ..., w_n) - D}{\sum_w N_{1+}(\bullet w_1, ..., w_{n-1}, w)}$$

- Again, three different values for D (D_1, D_2, D_{3+}), based on the count of the history $w_1, ..., w_{n-1}$

Formula for d for Lower Order N-Gram Models

- Probability mass set aside available for the d function

$$d(w_1, ..., w_{n-1}) = \frac{\sum_{i \in \{1,2,3+\}} D_i N_i(w_1, ..., w_{n-1} \bullet)}{\sum_{w_n} c(w_1, ..., w_n)}$$

Interpolated Back-Off

- Back-off models use only highest order n-gram
  - if sparse, not very reliable.
  - two different n-grams with same history occur once $\rightarrow$ same probability
  - one may be an outlier, the other under-represented in training

- To remedy this, always consider the lower-order back-off models

- Adapting the α function into interpolated α_I function by adding back-off

$$\begin{aligned}\alpha_I(w_n|w_1,...,w_{n-1}) = {} & \alpha(w_n|w_1,...,w_{n-1}) \\ & + d(w_1,...,w_{n-1})\; p_I(w_n|w_2,...,w_{n-1})\end{aligned}$$

- Note that d function needs to be adapted as well

Evaluation

Evaluation of smoothing methods:

Perplexity for language models trained on the Europarl corpus

Smoothing method	**bigram**	**trigram**	**4-gram**
Good-Turing	96.2	62.9	59.9
Witten-Bell	97.1	63.8	60.4
Modified Kneser-Ney	95.4	61.6	58.6
Interpolated Modified Kneser-Ney	94.5	59.3	54.0

Managing the Size of the Model

- Millions to billions of words are easy to get
  (trillions of English words available on the web)

- But: huge language models do not fit into RAM

Number of Unique N-Grams

Number of unique n-grams in Europarl corpus

29,501,088 tokens (words and punctuation)

Order	Unique n-grams	Singletons
unigram	86,700	33,447 (38.6%)
bigram	1,948,935	1,132,844 (58.1%)
trigram	8,092,798	6,022,286 (74.4%)
4-gram	15,303,847	13,081,621 (85.5%)
5-gram	19,882,175	18,324,577 (92.2%)

$\rightarrow$ remove singletons of higher order n-grams

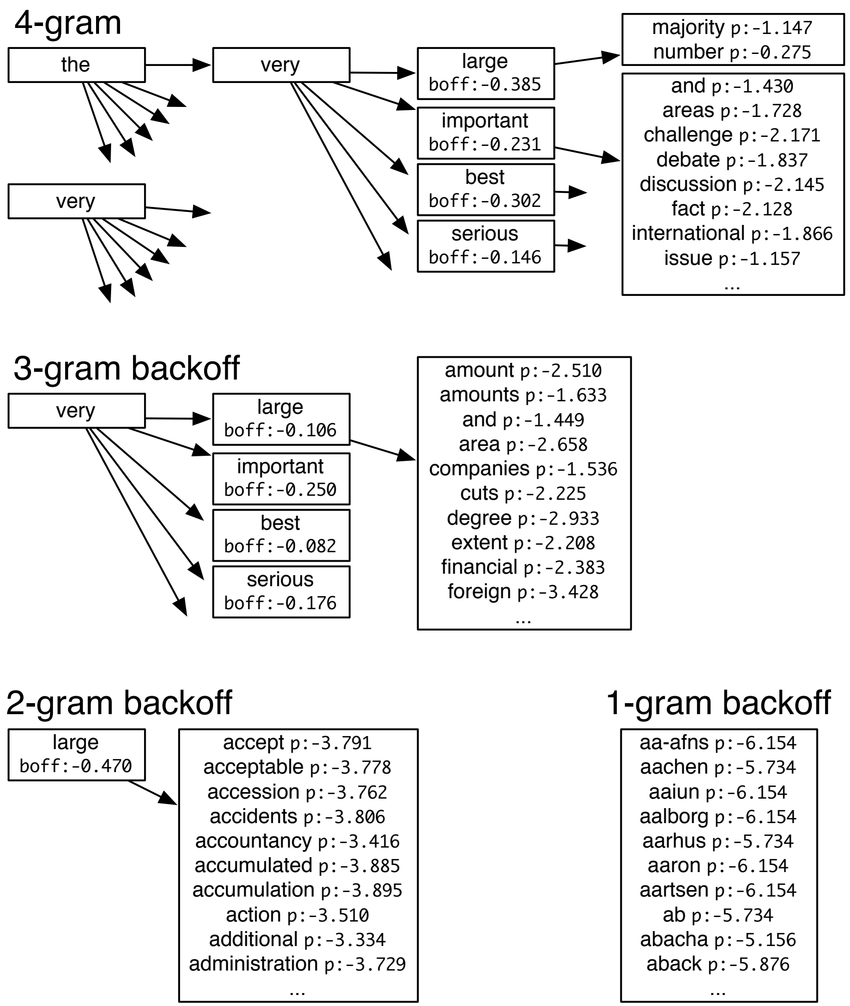
Estimation on Disk

- Language models too large to *build*
- What needs to be stored in RAM?
  - maximum likelihood estimation

$$p(w_n|w_1, ..., w_{n-1}) = \frac{\text{count}(w_1, ..., w_n)}{\text{count}(w_1, ..., w_{n-1})}$$

  - can be done separately for each history $w_1, ..., w_{n-1}$
- Keep data on disk
  - extract all n-grams into files on-disk
  - sort by history on disk
  - only keep n-grams with shared history in RAM
- Smoothing techniques may require additional statistics

Efficient Data Structures



- Need to store probabilities for
  - the very large majority
  - the very language number
- Both share history the very large

$\rightarrow$ no need to store history twice

$\rightarrow$ Trie

Fewer Bits to Store Probabilities

- Index for words
  - two bytes allow a vocabulary of $2^{16} = 65,536$ words, typically more needed
  - Huffman coding to use fewer bits for frequent words.

- Probabilities
  - typically stored in log format as floats (4 or 8 bytes)
  - quantization of probabilities to use even less memory, maybe just 4-8 bits

Reducing Vocabulary Size

- For instance: each number is treated as a separate token
- Replace them with a number token NUM
  - but: we want our language model to prefer

$$p_{\text{LM}}(\text{I pay 950.00 in May 2007}) > p_{\text{LM}}(\text{I pay 2007 in May 950.00})$$

  - not possible with number token

$$p_{\text{LM}}(\text{I pay } \textsc{num} \text{ in May } \textsc{num}) = p_{\text{LM}}(\text{I pay } \textsc{num} \text{ in May } \textsc{num})$$

- Replace each digit (with unique symbol, e.g., @ or 5), retain some distinctions

$$p_{\text{LM}}(\text{I pay 555.55 in May 5555}) > p_{\text{LM}}(\text{I pay 5555 in May 555.55})$$

Filtering Irrelevant N-Grams

- We use language model in decoding
  - we only produce English words in translation options
  - filter language model down to n-grams containing only those words
- Ratio of 5-grams needed to all 5-grams (by sentence length):