

# What was the problem?

- JOHN READ MOBY DICK •
- MARY READ A DIFFERENT BOOK •
- SHE READ A BOOK BY CHER •

$p(\bullet \text{ JOHN READ A BOOK } \bullet)$



$p(\bullet \text{ CHER READ A BOOK } \bullet)$

Does not happen  
in training.

$$p_2(\bullet \text{ CHER READ A BOOK } \bullet) = p(\text{CHER} | \bullet) p(\text{READ} | \text{CHER}) p(\text{A} | \text{READ}) p(\text{BOOK} | \text{A}) p(\bullet | \text{BOOK}) = 0$$

# Solution?

• JOHN READ MOBY DICK •  
• MARY READ A DIFFERENT BOOK •  
• SHE READ A BOOK BY CHER •

$p(\bullet \text{ JOHN READ A BOOK } \bullet)$



$p(\bullet \text{ CHER READ A BOOK } \bullet)$

Does not happen  
in training.

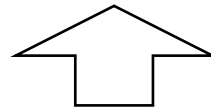
$$p_2(\bullet \text{ CHER READ A BOOK } \bullet) = p(\text{CHER} | \bullet) p(\text{READ} | \text{CHER}) p(\text{A} | \text{READ}) p(\text{BOOK} | \text{A}) p(\bullet | \text{BOOK}) = 0$$

$$p_1(\bullet \text{ CHER READ A BOOK } \bullet) = p(\bullet) p(\text{CHER}) p(\text{READ}) p(\text{A}) p(\text{BOOK}) p(\bullet)$$

Does happen!

# Linear Interpolation

$$P_{li}(w_i|w_{i-1}) = \lambda P(w_i) + (1 - \lambda)P(w_i|w_{i-1})$$



1-gram

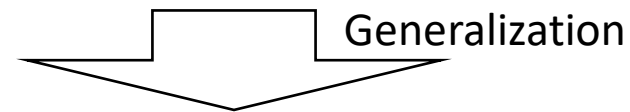


2-gram

See Jelinek-Merce

# Linear Interpolation

$$P_{li}(w_i|w_{i-1}) = \lambda P(w_i) + (1 - \lambda)P(w_i|w_{i-1})$$

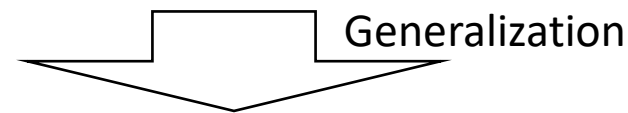


$$P(w_n|w_{n-k} \dots w_{n-1}) = \sum_{i=1}^k \lambda_i P_i(w_n|w_{n-i} \dots w_{n-1})$$

$$\sum_i \lambda_i = 1$$

# Linear Interpolation

$$P_l(w_i|w_{i-1}) = \lambda P(w_i) + (1 - \lambda)P(w_i|w_{i-1})$$



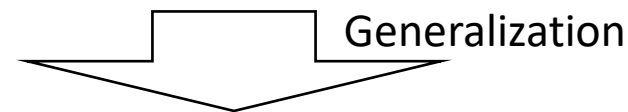
$$P_l(w_i|w_{i-n+1} \dots w_{i-1}) = \lambda_{w_{i-n+1} \dots w_{i-1}} P(w_i|w_{i-n+1} \dots w_{i-1}) \\ + (1 - \lambda_{w_{i-n+1} \dots w_{i-1}}) P_l(w_i|w_{i-n+2} \dots w_{i-1})$$

Context dependent interpolation!

Calculation: Expectation-maximization algorithm

# Linear Interpolation

$$P_l(w_i|w_{i-1}) = \lambda P(w_i) + (1 - \lambda)P(w_i|w_{i-1})$$



$$P_l(w_i|w_{i-n+1} \dots w_{i-1}) = \lambda_{w_{i-n+1} \dots w_{i-1}} P(w_i|w_{i-n+1} \dots w_{i-1}) \\ + (1 - \lambda_{w_{i-n+1} \dots w_{i-1}}) P_l(w_i|w_{i-n+2} \dots w_{i-1})$$

The formula can be applied recursively up to 0-gram.

# (Excursion: Voice Destination Entry)

Task: Enter address with speech in car

Problem:

- >>20million addresses in Germany
- Vocabulary size is >>1millionen

# (Excursion: Voice Destination Entry)

Task: Enter address with speech in car

Problem:

- >>20million addresses in Germany
  - Vocabulary size is >>1millionen
- 
- 0-gram: poor recognition ...
  - n-gram over „social data“: poor recognition ...
  - Solution?



# (Excursion: Voice Destination Entry)

Task: Enter address with speech in car

Problem: No geographical weighting of addresses

Probabilities resulting from  
databases, etc.



# (Excursion: Voice Destination Entry)

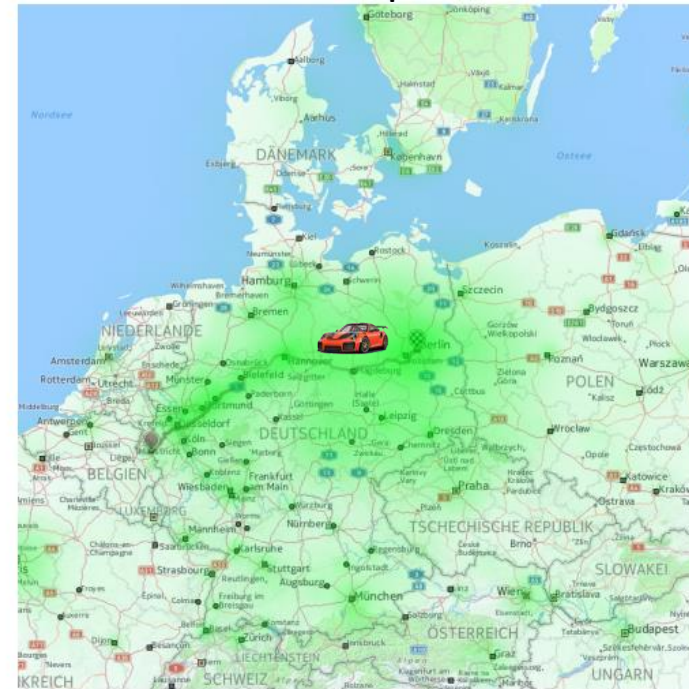
Task: Enter address with speech in car

Solution: Geographical weighting of addresses

Probabilities resulting from  
databases, etc.



Adjusted probabilities result from  
the user profile



# (Excursion: Voice Destination Entry)

Task: Enter address with speech in car

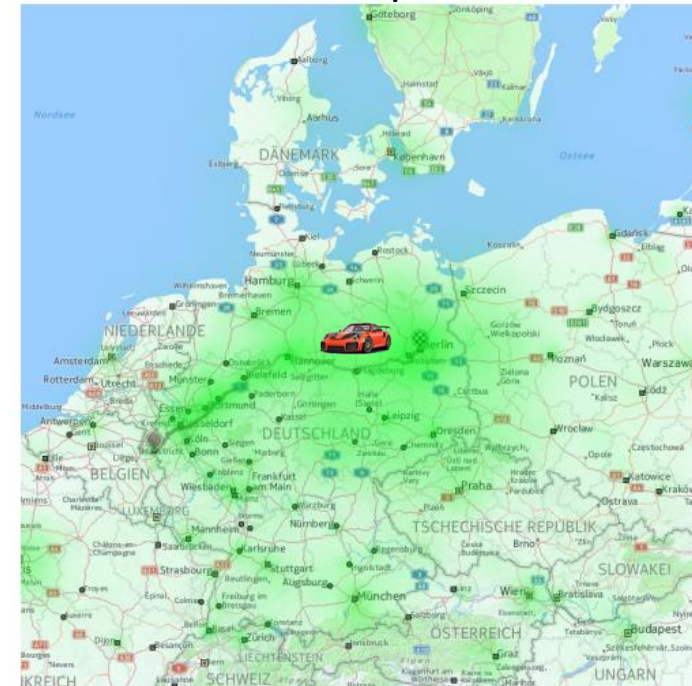
Solution: Geographical weighting of addresses

Adjusted probabilities result from  
the user profile

Estimated for each region

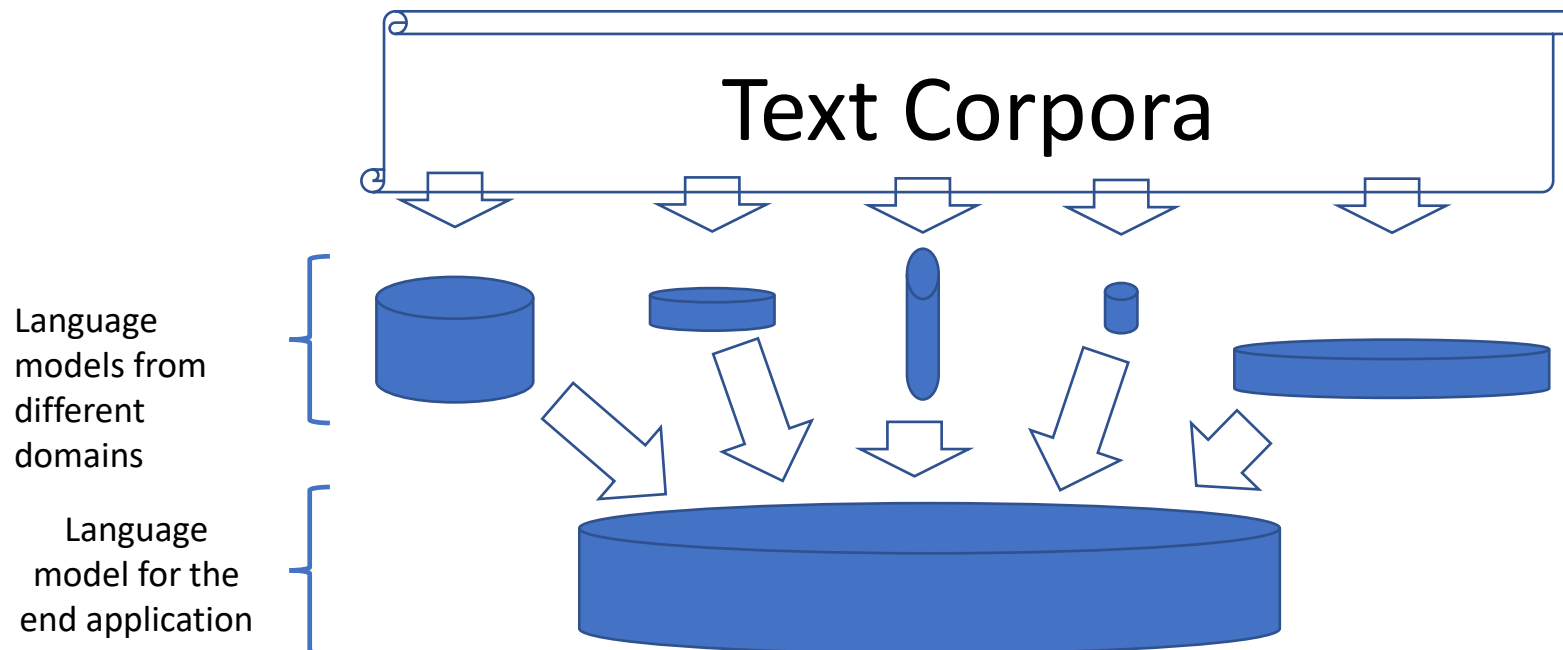
$$P(w_n | w_{n-k} \dots w_{n-1}) = \sum_{i=1}^k \lambda_i P_i(w_n | w_{n-i} \dots w_{n-1})$$

At the runtime of the recognition  
Adapted to the driver (user profile)



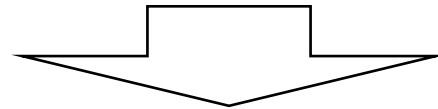
# Domain adaptation from language model

"Combine N-gram models (with under higher order) with other models (with under) lower order that were more likely to be observed in the training data."

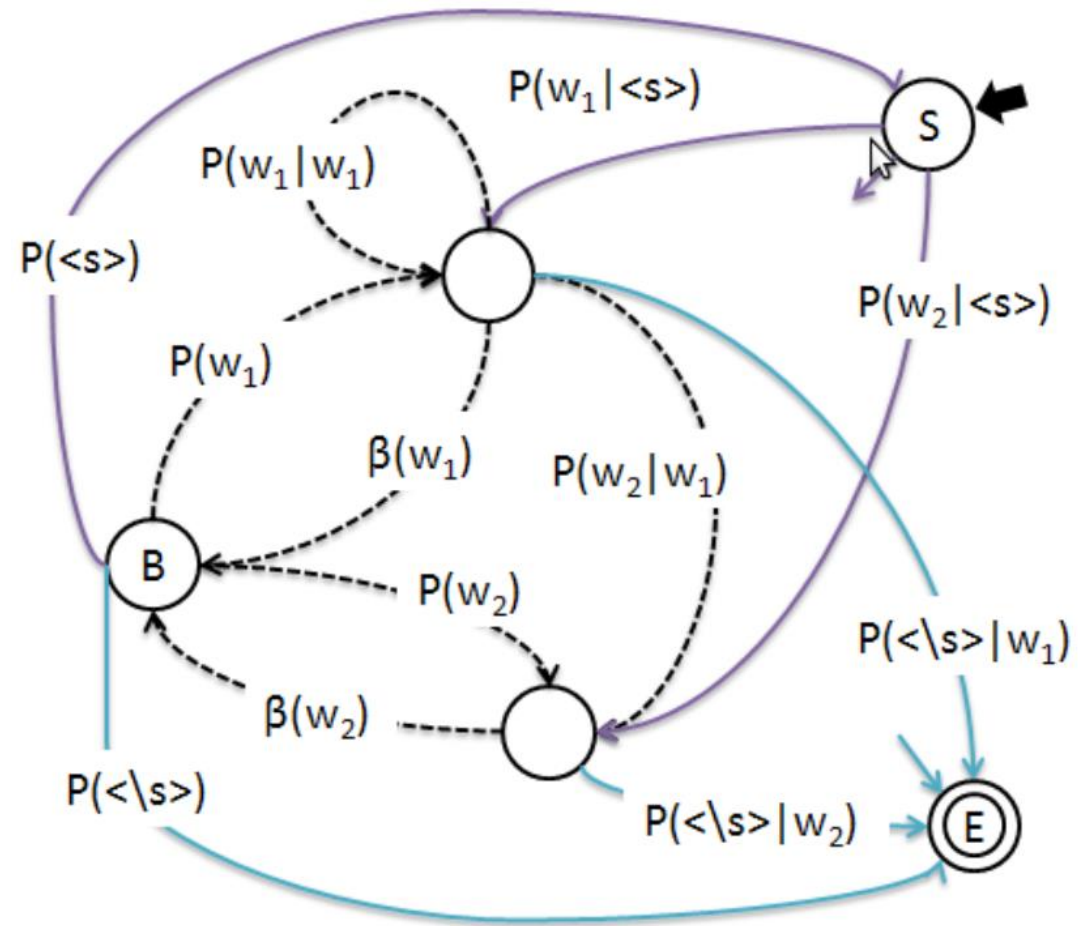


# Ethik

"The data used to train the language models might contain a 'bias', i.e. certain topics/expressions might be more represented than other topics/expressions. This 'bias' may not be - ethically - desired."

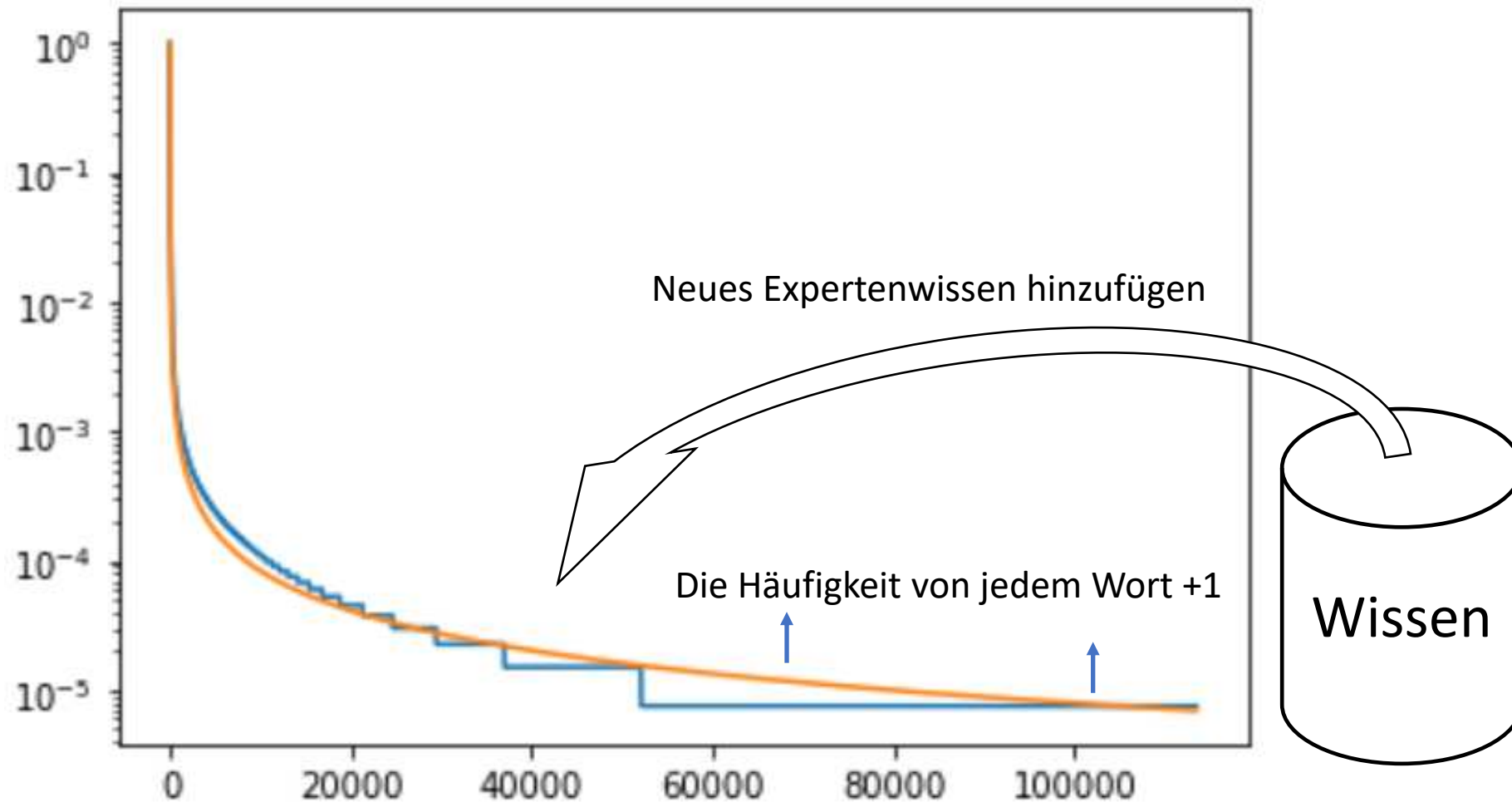


Always make sure that the data are representative! If necessary, decompose and interpolate.



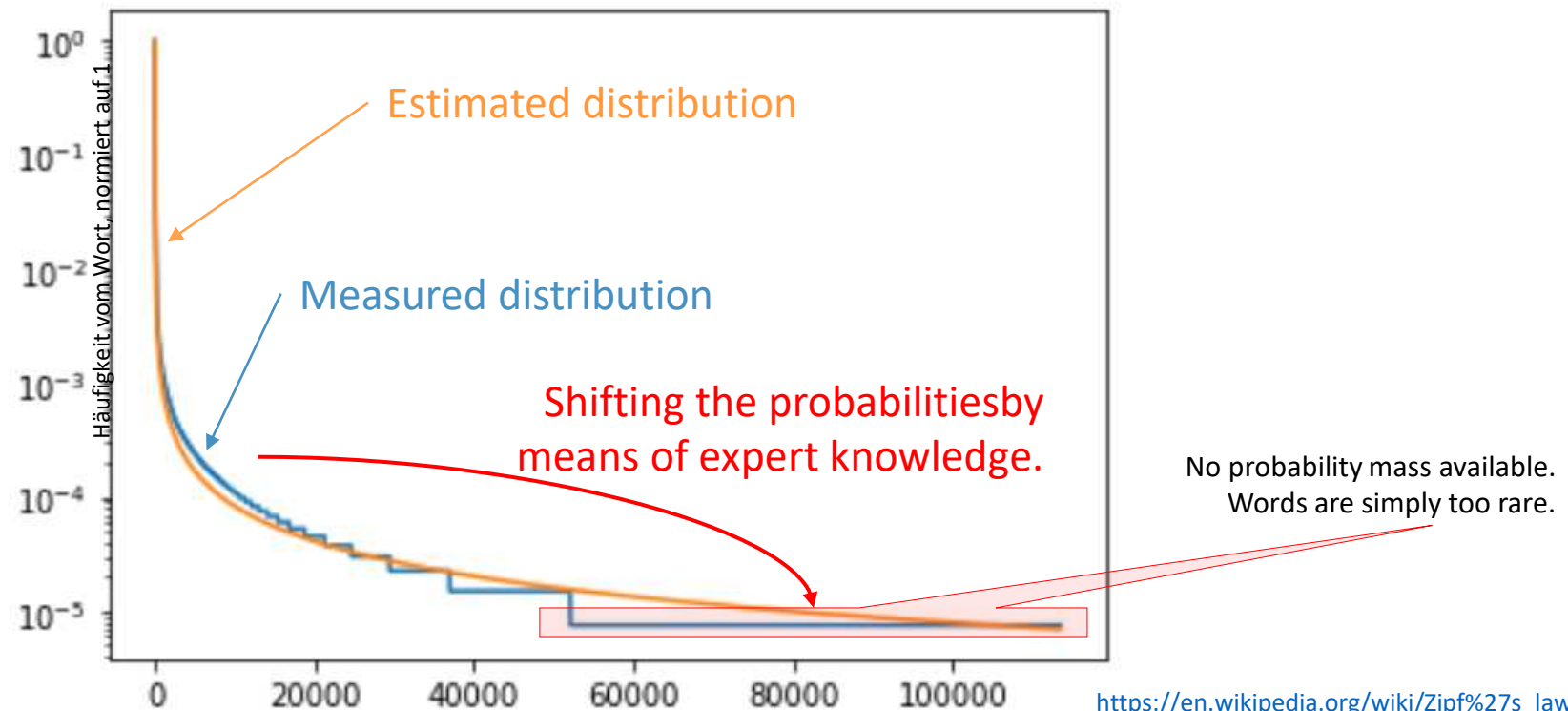


# What happened so far?



# Smoothing

"Subtract something from the probability mass of observed events and make it available for the unseen events."





# Smoothing: Absolute Discounting

„In absolute discounting, a constant very small absolute value is subtracted from the frequency of each observed N-gram, the discount. The probabilities of the N-grams are then determined based on the reduced frequencies. The sum of the discounts is distributed among the unseen N-grams as frequency values in the application”

# Smoothing: Absolute Discounting

„In absolute discounting, a constant very small absolute value is subtracted from the frequency of each observed N-gram, the discount. The probabilities of the N-grams are then determined based on the reduced frequencies. The sum of the discounts is distributed among the unseen N-grams as frequency values in the application”

## Training:

1000 different n-grams (types)

In total there are 10000 n-grams (tokens)

We set a discount of 0.1, i.e.  $1000 * 0.1 = 100$  frequencies can be reallocated

For a 3 times counted n-gram we get:  $3 - 0.1 / 10000 = 2.9 / 10000$

# Smoothing: Absolute Discounting

„In absolute discounting, a constant very small absolute value is subtracted from the frequency of each observed N-gram, the discount. The probabilities of the N-grams are then determined based on the reduced frequencies. The sum of the discounts is distributed among the unseen N-grams as frequency values in the application”

## Training:

1000 different n-grams (types)

In total there are 10000 n-grams (tokens)

We set a discount of 0.1, i.e.  $1000 * 0.1 = 100$  frequencies can be reallocated

For a 3 times counted n-gram we get:  $3 - 0.1 / 10000 = 2.9 / 10000$

## Test:

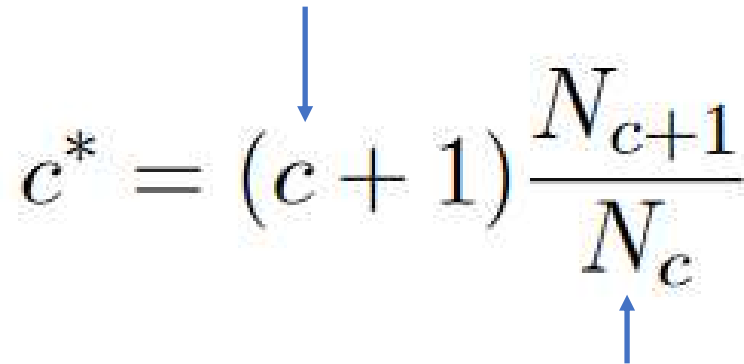
200 new n-grams not seen before.

Each n-gram is now assigned a  $(1000 * 0.1) / 200 = 100 / 200 = 0.5$  frequency

The probability with discounting for an unseen n-gram is:  $0.5 / 10000$

# Smoothing: Good-Turing Discounting

Number of n-gram tokens, e.g. "the car" occurs 500 times

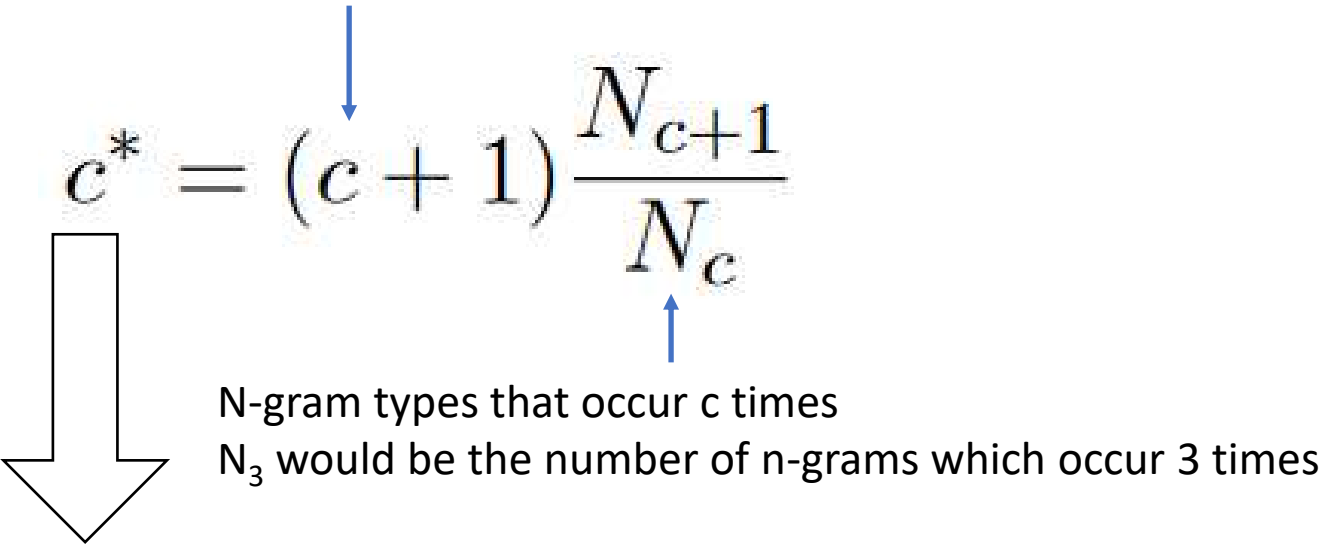

$$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$

N-gram types that occur c times

$N_3$  would be the number of n-grams which occur 3 times

# Smoothing: Good-Turing Discounting

Number of n-gram tokens, e.g. "the car" occurs 500 times


$$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$

N-gram types that occur c times

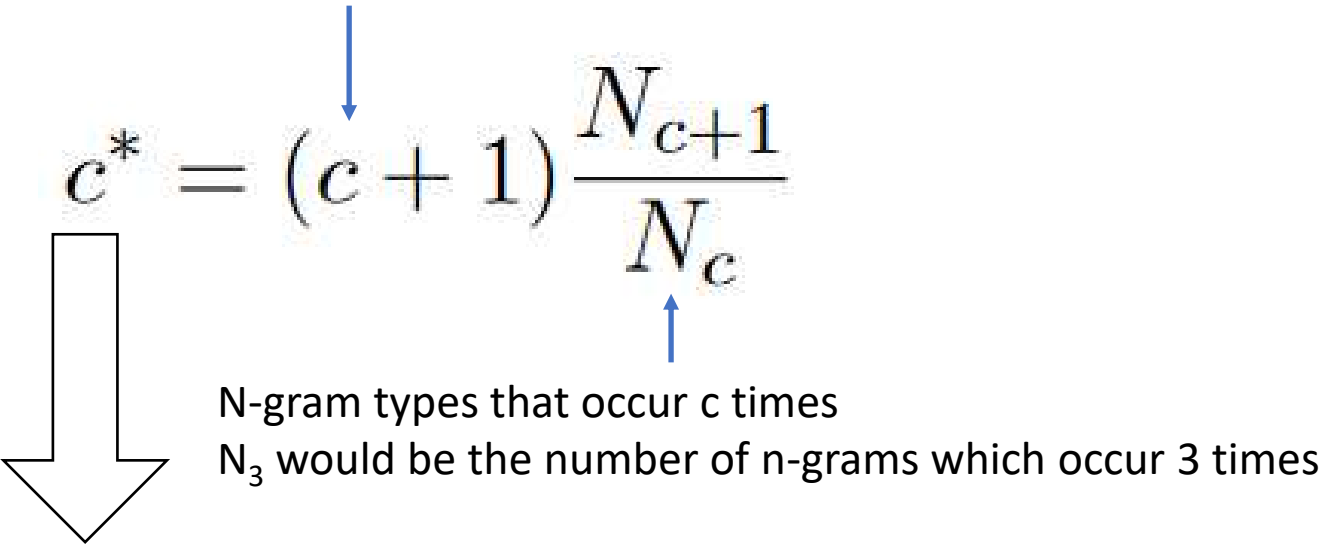
$N_3$  would be the number of n-grams which occur 3 times

$0^* > 0$  .... An n-gram that was not seen in the corpus is assigned a frequency of  $0^*$ . Overall, however, nothing changes:

$$\sum_{r=0}^{\infty} n_r r^* = \sum_{r=0}^{\infty} n_r (r + 1) \frac{n_{r+1}}{n_r} = \sum_{r=1}^{\infty} n_r r = N$$

# Smoothing: Good-Turing Discounting

Number of n-gram tokens, e.g. "the car" occurs 500 times


$$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$

N-gram types that occur c times

$N_3$  would be the number of n-grams which occur 3 times

$$P(w_i | w_{i-1}) = \frac{c^*(w_{i-1}w_i)}{c(w_{i-1})}$$

# Smoothing: Good-Turing Discounting

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

# Smoothing: Good-Turing Discounting

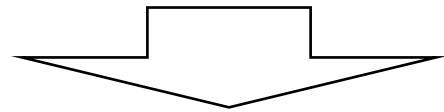
Count	Count of counts	Adjusted count	Test count
$r$	$N_r$	$r^*$	$t$
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927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



# Smoothing: Good-Turing in application

As always... usually  $k=5$

$$c^* = \begin{cases} c & : c > k \\ \frac{(c+1) \frac{N_{c+1}}{N_c} - c \frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} & : \text{else} \end{cases}$$



The frequency is corrected upwards

$$P(w_i | w_{i-1}) = \frac{c^*(w_{i-1}w_i)}{c(w_{i-1})}$$

# Smoothing: Good-Turing Discounting

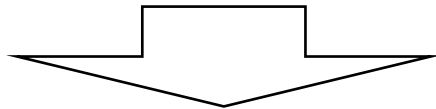
- Assumes that if  $N_i > 0$ , then also  $N_{i-1} > 0$
- Uses discounting even at high frequencies

# Smoothing: Good-Turing Discounting

- Assumes that if  $N_i > 0$ , then also  $N_{i-1} > 0$
- Uses discounting even at high frequencies
- Assigns equal probability to all "unseen n-grams".:  
Not seen during 3-gram training:
  - Scottish beer drinkers
  - Scottish beer eaters} would be assigned equal probability!

# Smoothing: Good-Turing Discounting

- Assumes that if  $N_i > 0$ , then also  $N_{i-1} > 0$
  - Uses discounting even at high frequencies
  - Assigns equal probability to all "unseen n-grams".:  
Not seen during 3-gram training:
    - Scottish beer drinkers
    - Scottish beer eaters
- } would be assigned equal probability!



- Interpolation
- Back-off

# Katz's Back-Off Language Models

“A generative n-gram language model that estimates the conditional probability of a word given its history in the n-gram.”

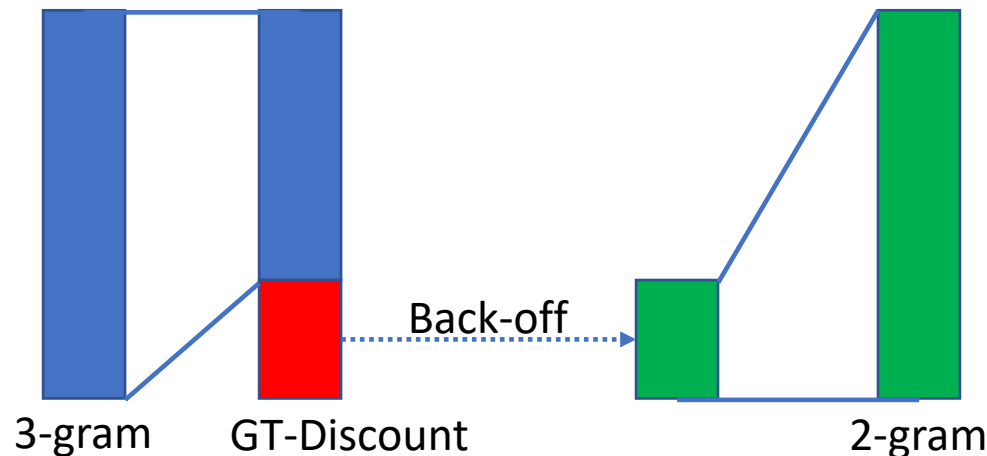
# Katz's Back-Off

- based on Good-Turing Discounting
- Discount n-gram probabilities

# Katz's Back-Off

- based on Good-Turing Discounting
- Discount n-gram probabilities
- Katz Back-off:

Discount is not uniformly distributed, but dependent on the (n-1)-gram



# Back-Off Language Models

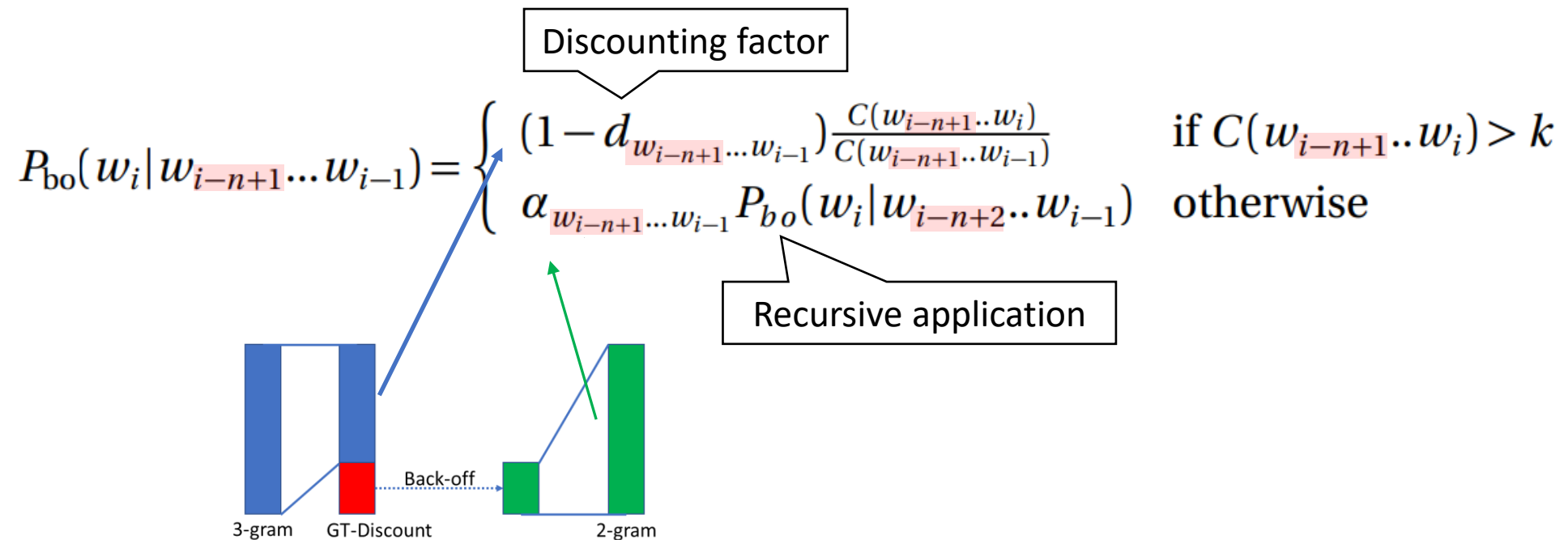
„Use the count of things we’ve seen once to help estimate the count of things we’ve never seen.“

$$P_{\text{bo}}(w_i | w_{i-n+1} \dots w_{i-1}) = \begin{cases} (1 - d_{\overbrace{w_{i-n+1} \dots w_{i-1}}^{\text{n-gram}}}) \frac{C(w_{i-n+1} \dots w_i)}{C(w_{i-n+1} \dots w_{i-1})} & \text{if } C(w_{i-n+1} \dots w_i) > k \\ \dots & \dots \end{cases}$$



# Back-Off Language Models

„Use the count of things we’ve seen once to help estimate the count of things we’ve never seen.“



# Back-Off Language Models

„Use the count of things we’ve seen once to help estimate the count of things we’ve never seen.“

$$P_{\text{bo}}(w_i | w_{i-n+1} \dots w_{i-1}) = \begin{cases} (1 - d_{w_{i-n+1} \dots w_{i-1}}) \frac{C(w_{i-n+1} \dots w_i)}{C(w_{i-n+1} \dots w_{i-1})} & \text{if } C(w_{i-n+1} \dots w_i) > k \\ \alpha_{w_{i-n+1} \dots w_{i-1}} P_{\text{bo}}(w_i | w_{i-n+2} \dots w_{i-1}) & \text{otherwise} \end{cases}$$

Discounting factor

Back-off weight

Recursive application

Yields correct probabilities

$$P(a_i) \geq 0$$

$$0 \leq P(a_i) \leq 1, P \in \mathbb{R}$$

$$P(a_0) + \dots + P(a_n) = 1$$

# Kneser-Ney Language Models

“The lower-order model is significant only when count is small or zero in the higher-order model, and so should be optimized for that purpose”

# Kneser-Ney Language Models

„The lower-order model is significant only when count is small or zero in the higher-order model, and so should be optimized for that purpose“

# Kneser-Ney Language Models

„The lower-order model is significant only when count is small or zero in the higher-order model, and so should be optimized for that purpose“

Example:

„San Francisco“ appears frequently.

„Francisco“ appears only right after „San“.

=> Assign „Francisco“ a lower 1-gram probability, since it co-occurs only with „San“ in the 2-gram: „San Francisco“.

- **Maximum-Likelihood-Estimation**
- **Interpolating different Language Models:**
  - Of different order
  - From different data subsets
- **Additive smoothing**
  - Laplace, add-one oder Lidstone & Jeffreys smoothing
- **Absolut & Good-Turing Discounting**
- **Back-off**
  - Mini introduction: Katz method
  - Mini introduction : Kneser-Ney method

# Class-based Language Models

“Let’s put it all in Classes”

# Class-based SLM

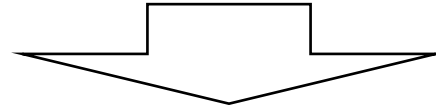
$$P(w_t | \text{context})$$

„context“ can mean lots of things...



# Class-based SLM, word classes

$$P(w_t | \text{context})$$



$$P(w_i | C_{i-n+1} \dots C_{i-1}) = P(w_i | C_i) P(C_i | C_{i-n+1} \dots C_{i-1})$$

Every word could be assigned a class, z.B.

go, walk, drink, eat => verb

car, boat, plain, lighter => noun

or

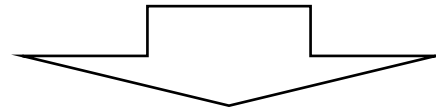
Munir, Marie, Josef, Sören => names

Munir, Sören => professors

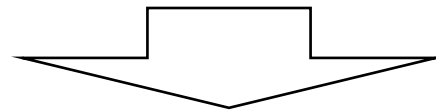
Munir => Spoken and Natural Language Understanding

# Class-based SLM, word classes

$$P(w_t | \text{context})$$



$$P(w_i | C_{i-n+1} \dots C_{i-1}) = P(w_i | C_i) P(C_i | C_{i-n+1} \dots C_{i-1})$$



$$P(\underline{w}) = \sum_{c_1 \dots c_n} \prod_{i=1} P(w_i | c_i) P(C_i | C_{i-n+1} \dots C_{i-1})$$

Every word could be assigned a class, z.B.

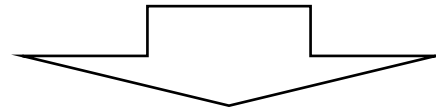
Munir, Marie, Josef, Sören => names

Munir, Sören => professors

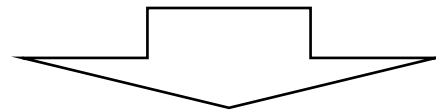
Munir => Spoken and Natural Language Understanding

# Class-based SLM, word classes

$$P(w_t | \text{context})$$



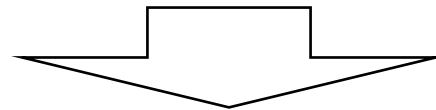
$$P(w_i | C_{i-n+1} \dots C_{i-1}) = P(w_i | C_i) P(C_i | C_{i-n+1} \dots C_{i-1})$$



Each word can be assigned exactly one class, e.g.

go, walk, drink, eat => verb

car, boat, plain, lighter => noun



$$P(\underline{w}) = \prod_{i=1} P(w_i | c_i) P(C_i | C_{i-n+1} \dots C_{i-1})$$

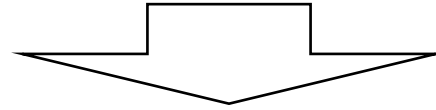
# Class-based SLM

$$P(w_t | \text{context})$$

„context“ can be anything...

# Class-based SLM, context classes

$$P(w_t | \text{context})$$



$$P(\underline{w}) = \prod_{i=1} P(w_i | E(w_1 \dots w_{i-1}))$$

Equivalence classes of  
„previous“ words

# Questioning the n-gram

“Out-Of-Box Thinking”

# Example 0

This lecture is ...

# Example 0 (Auflösung)

This lecture is

exciting

super

ingenious

good

interesting

informative

commendable

the best

worth a recommendation



# Example 1

Rlealy crdue! Ainccordg to a uersinivty sudty, it deson't meattr waht odrer the ltteers aer in a wrod, eth olny tinhg taht maertts is taht the frsit and lsat leettrs are in the crroect pootisin.

The rest can be tatol nseoenns, but you can sltil raed it wouthit any pmroble. Tihs is buscaee we dn'ot raed ecah ltteer indudilvialy, but reizcogne the wrdo as a wolhe. Rlealy crdue! It ralley wrkos! Adn th'ats why we go to shoocl for yares!

# Example 1 (plain text)

Really crude! According to a university study, it doesn't matter what order the letters are in a word, the only thing that matters is that the first and last letters are in the correct position.

The rest can be total nonsense, but you can still read it without any problems. This is because we don't read each letter individually, but recognize the word as a whole. Really crude! It really works! And that's why we go to school for years!

# Example 2

7h15 m355463 5h0w5 y0u wh47 6r347 f3475 0ur br41n 15 c4p4bl3 0f.  
47 7h3 b361nn1n6, 17 w45 c3r741nly d1ff1cul7 70 r34d 7h15, bu7 by  
n0w y0u c4n pr0b4bly r34d 17 qu173 w3ll w17h0u7 r34lly 57r41n1n6  
y0ur53lf. 7h15 15 wh47 y0ur br41n d035 w17h 175 3n0rm0u5  
l34rn1n6 4b1l17y. 1mpr3551v3, 15n'7 17? y0u 4r3 w3lc0m3 70 c0py  
7h15 1f y0u w4n7 70 1n5p1r3 07h3r5 w17h 17.

## Example 2 (plain text)

this message shows you what great feats our brain is capable of. At the beginning, it was certainly difficult to read this, but by now you can probably read it quite well without really straining yourself. This is what your brain does with its enormous learning ability. Impressive, isn't it? You are welcome to copy this if you want to inspire others with it.

# Generalization n-gram

1-gram:	$P(w_i)$
left 2-gram:	$P(w_{i-1}w_i)$
Right 2-gram:	$P(w_iw_{i+1})$
left 3-gram:	$P(w_{i-2}w_{i-1}w_i)$
Right 3-gram:	$P(w_iw_{i+1}w_{i+2})$
Middle 3-gram:	$P(w_{i-1}w_iw_{i+1})$
Skip-gram:	$P(w_{i-1}w_{i+1})$
...	

# Advanced Language Modelling

“Let’s adapt.”

# Cache Language Model

Language model used in recognition

$$P_c(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1 - \lambda)P(w_i|w_{i-1})$$

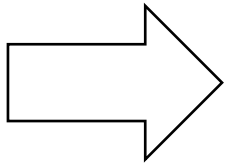
Cache LM estimated from recognized  
text during runtime

# Cache Language Model

Language model used in recognition

$$P_c(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1 - \lambda)P(w_i|w_{i-1})$$

Cache LM estimated from recognized  
text during runtime



good idea, but hardly works in practice... why?

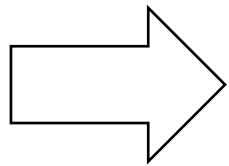


# Cache Language Model

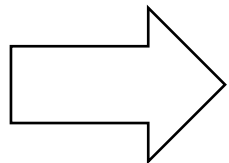
Language model used in recognition

$$P_c(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1 - \lambda)P(w_i|w_{i-1})$$

Cache LM estimated from recognized  
text during runtime



good idea, but hardly works in practice... why?



- very limited data available to estimate a LM, even for a uni-gram
- Idea lags a bit; Automatic Speech Recognition errors; adaptation to errors useful?
- Cache LM is trained on keyboard data

# Topic Adaptive Language Model

Language model used in recognition

$$P_c(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1 - \lambda)P(w_i|w_{i-1})$$

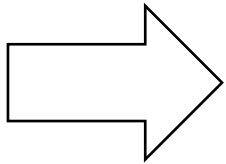
LM for a specific topic, topic is identified during runtime

# Topic Adaptive Language Model

Language model used in recognition

$$P_c(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1 - \lambda)P(w_i|w_{i-1})$$

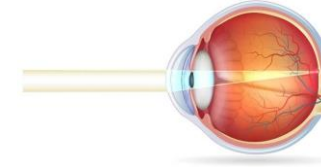
LM for a specific topic, topic is identified during runtime



Very good idea, works well in practice.... if:

- enough has been said to recognize the topic
- the topics are similar, otherwise recognition errors are too high...
- the topics are detected in a different way

# Fixation and saccades

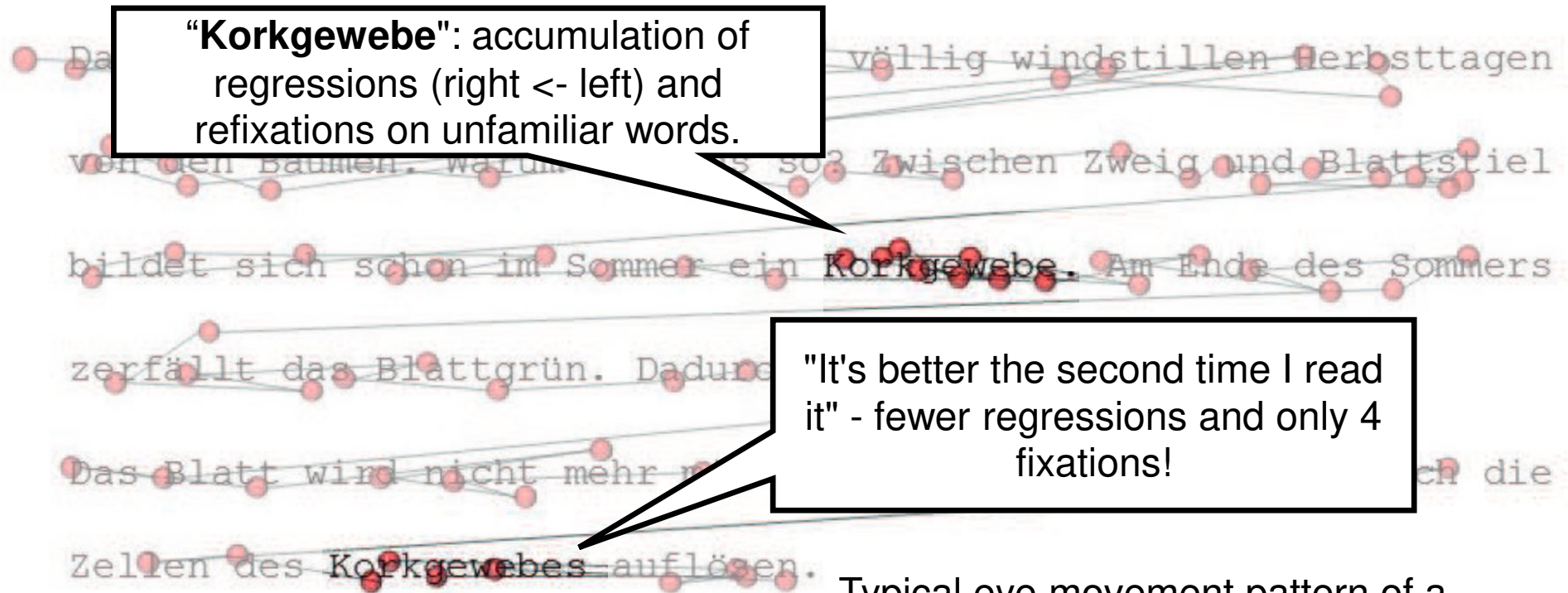


Das Laub im Herbst fällt auch an völlig windstillen Herbsttagen von den Bäumen. Warum ist das so? Zwischen Zweig und Blattstiel bildet sich schon im Sommer ein Korkgewebe. Am Ende des Sommers zerfällt das Blattgrün. Dadurch verfärbt sich das Laub. Das Blatt wird nicht mehr mit Nährstoffen versorgt, da sich die Zellen des Korkgewebes auflösen.

Fixation

Typical eye movement pattern of a fourth-grade student reading a relatively difficult page of text.

# Fixation and saccades



Typical eye movement pattern of a fourth-grade student reading a relatively difficult page of text.

# Further Language Models

“Learn distributed representations to  
reduce the impact of the curse of  
dimensionality”

# Overview

- Maximum Entropy Models
- Decision trees
- Grammars
- Stochastic Grammars
- ...

} Very good models, but cumbersome.

} Always a welcome sight, hardly ever used in practice.

- **Neural networks**

} Will be discussed!

# Comparison of LMs

Model	PPL
3-gram with Good-Turing smoothing (GT3)	165.2
5-gram with Kneser-Ney smoothing (KN5)	141.2
5-gram with Kneser-Ney smoothing + cache	125.7
Maximum entropy model	142.1
Random clusterings LM	170.1
Random forest LM	131.9
Structured LM	146.1
Within and across sentence boundary LM	116.6
Log-bilinear LM	144.5
Feedforward NNLM	140.2
Syntactical NNLM	131.3
Combination of static RNNLMs	102.1
Combination of adaptive RNNLMs	101.0



# Interpolated LMs

Model	Weight	PPL
3-gram with Good-Turing smoothing (GT3)	0	165.2
5-gram with Kneser-Ney smoothing (KN5)	0	141.2
5-gram with Kneser-Ney smoothing + cache	0.0792	125.7
Maximum entropy model	0	142.1
Random clusterings LM	0	170.1
Random forest LM	0.1057	131.9
Structured LM	0.0196	146.1
Within and across sentence boundary LM	0.0838	116.6
Log-bilinear LM	0	144.5
Feedforward NNLM	0	140.2
Syntactical NNLM	0.0828	131.3
Combination of static RNNLMs	0.3231	102.1
Combination of adaptive RNNLMs	0.3058	101.0
ALL	1	83.5