Lecture 4: Generalizing to unseen data: Smoothing

Jelle Zuidema
ILLC, Universiteit van Amsterdam

(based on slides from Tejaswini Deoskar, Yoav Seginer and Khalil Sima'an)

Taalmodellen 2011, BSc AI

Recap

- Relation between language model and next word prediction
- Markov assumptions: time invariance & limited history
- Extracting ngrams from corpora
- Relation between ngrams, HMMs and PCFGs

Generalization & Zipf's law

- Learning $\approx$ Generalization to unseen data
- Zipf's law: the fat head and the long tail

Smoothing (I)

- Naive add-λ
- Good-Turing smoothing

Alan Turing

Smoothing (II)

- Katz back-off
- Jelinek-Mercer
- from ngrams to PCFGs

Recap: language models and next word prediction

Language/prediction models: $P(w_0, w_1, \ldots, w_m)$

Chain Rule $P(w_0, w_1, \ldots, w_m) = P(w_0) \prod_{i=1}^{m} P(w_i | w_0, \ldots w_{i-1})$

Approximation (n-grams)

$$P(w_i \mid w_1, \ldots w_{i-k} \ldots, w_{i-1}) \approx P(w_i \mid w_{i-k}, \ldots, w_{i-1})$$

Markov models An $(n-1)^{th}$ order Markov model (n-gram):

0-order Markov	$P(w_0, w_1, \ldots, w_m) = P(w_0) \prod_{i=1}^{m} P(w_i)$
1-order Markov	$P(w_0, w_1, \ldots, w_m) = P(w_0) \prod_{i=1}^{m} P(w_i \vert w_{i-1})$
2-order Markov	$P(w_0, w_1, \ldots, w_m) =$ $P(w_0) \prod_{i=1}^{m} P(w_i \vert w_{i-2}, w_{i-1})$

Recap: Markov Assumptions

The Markov assumptions:

Time invariance: independent and identical trials!

Limited history: There is a fixed finite k such that for all w_1^{i+1}:

$$P(w_{i+1}|w_1, \ldots, w_i) \approx P(w_{i+1}|w_{i-k}, \ldots, w_i)$$

Recap: Estimation

Relative Frequency Estimate

Probabilities of an ngram (word|history) are estimated from frequency of wordin a corpus relative to other words with same history:

$$P(w_i \mid w_{i-k}, \ldots, w_{i-1}) = \frac{Count(w_{i-k}, \ldots, w_{i-1}, w_i)}{\sum_{w \in V} Count(w_{i-k}, \ldots, w_{i-1},\ w)}$$

Recap: Estimation

Relative Frequency Estimate

Probabilities of an ngram (word|history) are estimated from frequency of wordin a corpus relative to other words with same history:

$$P(w_i \mid w_{i-k}, \ldots, w_{i-1}) = \frac{Count(w_{i-k}, \ldots, w_{i-1}, w_i)}{\sum_{w \in V} Count(w_{i-k}, \ldots, w_{i-1},\ w)}$$

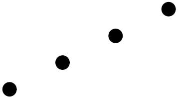
Addition of START and STOP

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{i=n+1} P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$

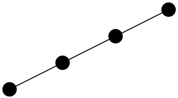
where $w_j =< s >\ \ (START)$ for $j \leq 0$ en $w_{n+1} =< /s >\ \ (STOP)$

$$\sum_{w \in V \cup \{STOP\}} Count(w_{i-k}, \ldots, w_{i-1},\ w) = Count(w_{i-k}, \ldots, w_{i-1})$$

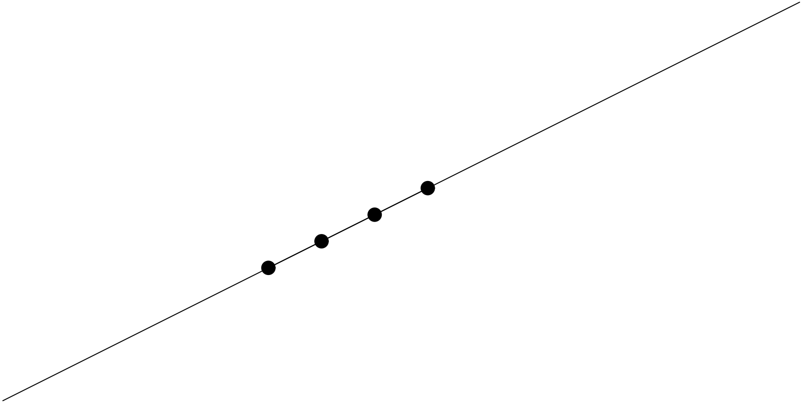
Learning from data



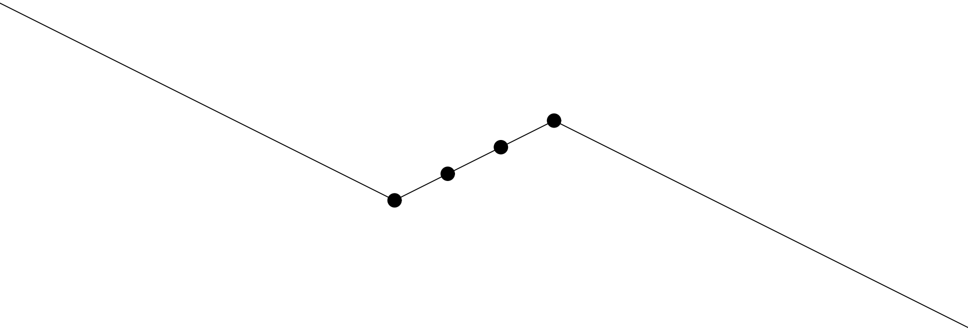
Interpolation



Extrapolation



Extrapolation



Learning from data

Easy, if

- Simple relation between input and output

Learning from data

Easy, if

- Simple relation between input and output
- Input and output have few dimensions and few possible values

Learning from data

Easy, if

- Simple relation between input and output
- Input and output have few dimensions and few possible values
- Training data is a good sample of test domain

Learning from data

Easy, if

- Simple relation between input and output
- Input and output have few dimensions and few possible values
- Training data is a good sample of test domain

Learning from data

Easy, if

- Simple relation between input and output
- Input and output have few dimensions and few possible values
- Training data is a good sample of test domain

NLP: assigning probabilities, syntactic analyses or meanings to sequences of words

- Complicated relationship between a sequence of words and its probability, grammaticality or meaning (cf. "arbitrariness", "compositionality", "recursion")

Learning from data

Easy, if

- Simple relation between input and output
- Input and output have few dimensions and few possible values
- Training data is a good sample of test domain

NLP: assigning probabilities, syntactic analyses or meanings to sequences of words

- Complicated relationship between a sequence of words and its probability, grammaticality or meaning (cf. "arbitrariness", "compositionality", "recursion")
- Very many basic units (rules, words, constructions)

Learning from data

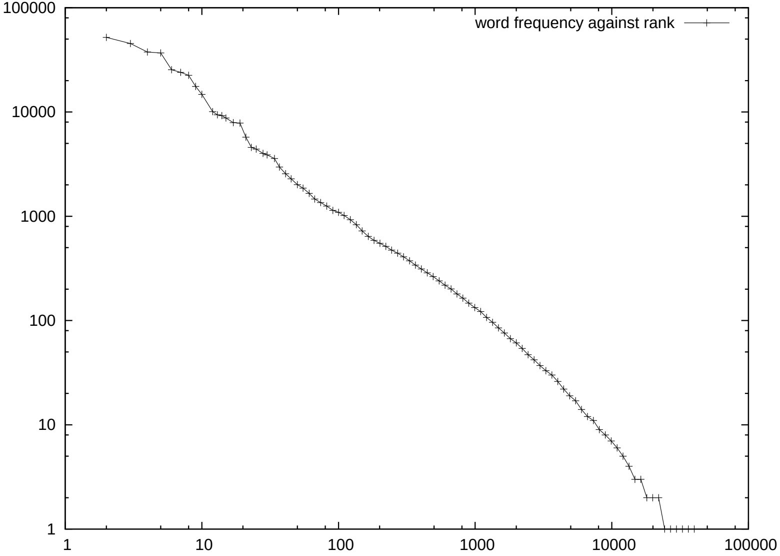
Easy, if

- Simple relation between input and output
- Input and output have few dimensions and few possible values
- Training data is a good sample of test domain

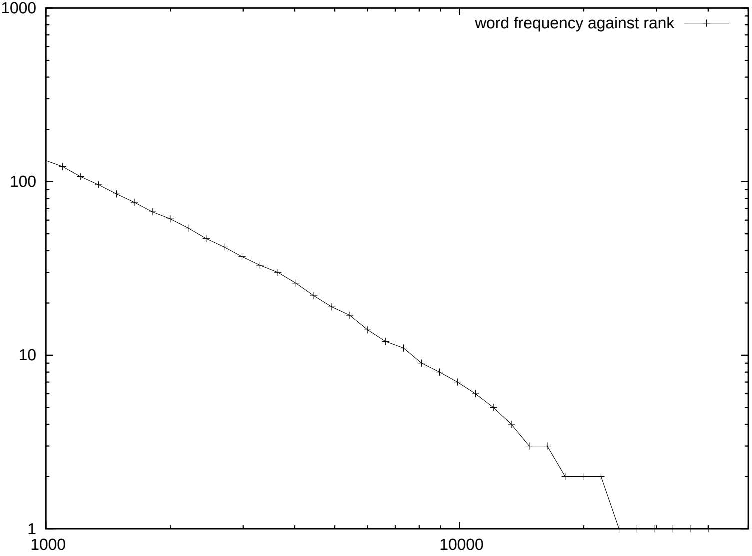
NLP: assigning probabilities, syntactic analyses or meanings to sequences of words

- Complicated relationship between a sequence of words and its probability, grammaticality or meaning (cf. "arbitrariness", "compositionality", "recursion")
- Very many basic units (rules, words, constructions)
- Very many rare events (cf. Zipf's law), poor generalization across domains, inherently dynamic.

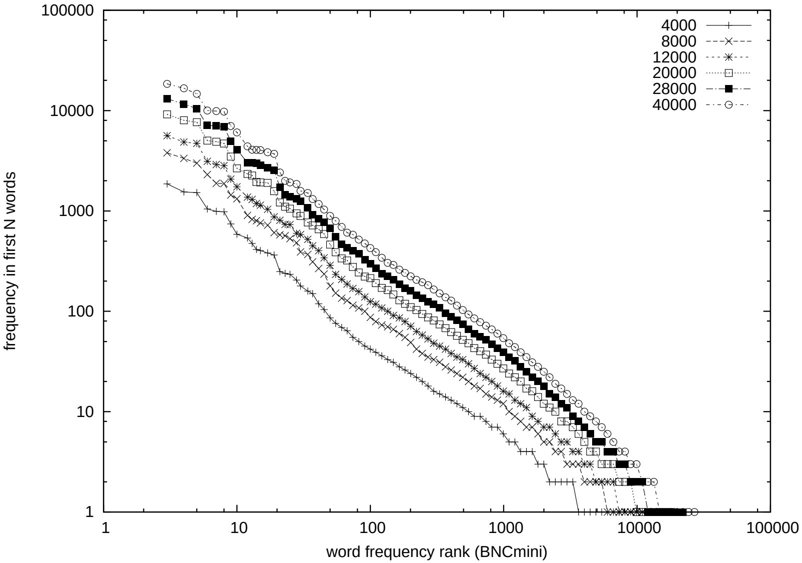
Zipf distribution (revisited)



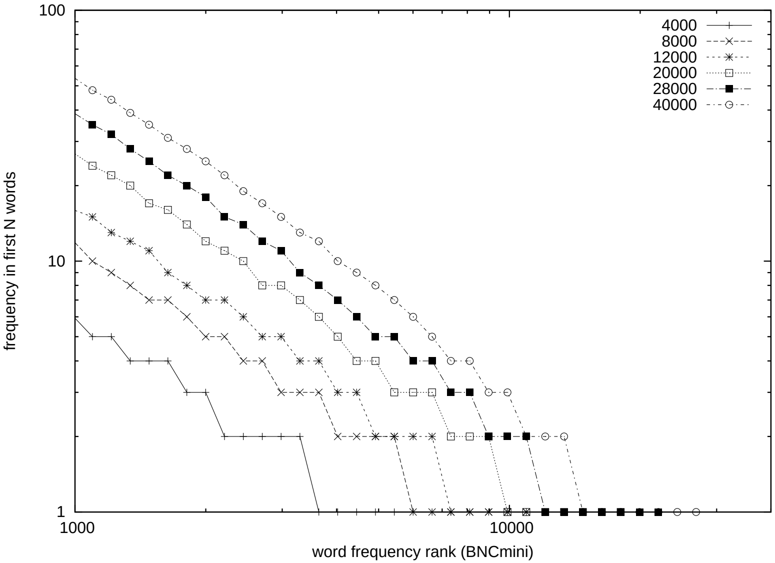
Vocabulary size: 43729? Unobserved true words?



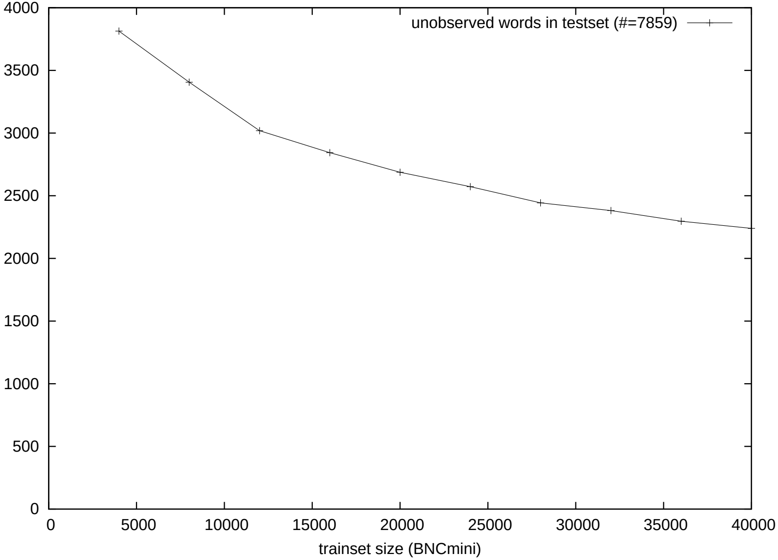
Word-frequency distributions in corpora of different sizes



Estimates of vocabulary size depend on corpus size!



unobserved words decreases only slowly with corpus size!



Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) = 0$

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) = 0$

Data: text does not contain bigrams $\langle START, Jerry\rangle$ and $\langle Jerry, sees\rangle$

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) = 0$

Data: text does not contain bigrams $\langle START, Jerry\rangle$ and $\langle Jerry, sees\rangle$

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) \;=\; 0$

Data: text does not contain bigrams $\langle START, Jerry \rangle$ and $\langle Jerry, sees \rangle$

Zeros: are called "unseen events",

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) \; = \; 0$

Data: text does not contain bigrams $\langle START, Jerry \rangle$ and $\langle Jerry, sees \rangle$

Zeros: are called "unseen events",

Question: the above sentence should have a non-zero probability, how do we estimate its probability?

Zero counts

Model: we train a bigram model on Data: $P(w_i|w_{i-1})$, e.g. "the cat sees the mouse", ..., "Tom sees Jerry", "the mouse sees Jerry"

Problem: $P(\text{"Jerry sees the mouse"}) = 0$

Data: text does not contain bigrams $\langle START, Jerry \rangle$ and $\langle Jerry, sees \rangle$

Zeros: are called "unseen events",

Question: the above sentence should have a non-zero probability, how do we estimate its probability?

- Why are zero's a problem for language models?

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing
- The problem will get worse as our language models get more informed by adding linguistic knowledge (it is time-consuming to annotate data)

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing
- The problem will get worse as our language models get more informed by adding linguistic knowledge (it is time-consuming to annotate data)
  - For e.g., in case of n-gram models, sparse-data problem is worse as n increases.

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing
- The problem will get worse as our language models get more informed by adding linguistic knowledge (it is time-consuming to annotate data)
  - For e.g., in case of n-gram models, sparse-data problem is worse as n increases.
- Symptom for another problem: Maximum-Likelihood counts might be either too high or too low, which implies a suboptimal model : Model does not *generalize*.

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing
- The problem will get worse as our language models get more informed by adding linguistic knowledge (it is time-consuming to annotate data)
  - For e.g., in case of n-gram models, sparse-data problem is worse as n increases.
- Symptom for another problem: Maximum-Likelihood counts might be either too high or too low, which implies a suboptimal model : Model does not *generalize*.

Why are zero probabilities a problem?

- Lack of robustness: if our estimate of the probability of some sentence in the input is zero, then we can do nothing with this sentence in further processing
- The problem will get worse as our language models get more informed by adding linguistic knowledge (it is time-consuming to annotate data)
  - For e.g., in case of n-gram models, sparse-data problem is worse as n increases.
- Symptom for another problem: Maximum-Likelihood counts might be either too high or too low, which implies a suboptimal model : Model does not *generalize*.

So: we will need a general solution for this.

More data: Does that solve the problem completely?

There are always events that will be missing.

Zipf's law: an empirical observation about word frequency distributions.

$freq(e)$	the frequency of e in naturally occuring data
$rank(e)$	the rank of e in the list ordered by frequency

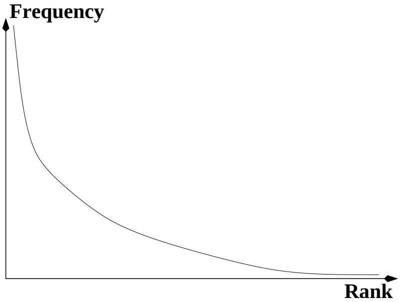
Zipf's law: There is a constant K such that for all e

$$freq(e) \times rank(e) = K$$

Example: An event of the $100,000^{th}$ rank occurs $10,000$ times less often than an event of the 10^{th} rank, i.e. $freq(e_{100000}) = \frac{1}{10{,}000} \times freq(e_{10})$

Hence: Very many infrequent words and only few frequent ones in any text.

Zipf's law



- Very many infrequent words and a few frequent ones in any text.
- Having a resonably sized corpus is important, but we always need smoothing.
- Smoothing technique used has a large effect on the performance of any NLP system.

Smoothing techniques for Markov Models

General Idea: Discount and redistribute

- Reserve mass from seen events in order to give to unseen events.

Smoothing techniques for Markov Models

General Idea: Discount and redistribute

- Reserve mass from seen events in order to give to unseen events.

But... the devil is in the details!

- How to discount mass in a proper way? (how much is enough)

Smoothing techniques for Markov Models

General Idea: Discount and redistribute

- Reserve mass from seen events in order to give to unseen events.

But... the devil is in the details!

- How to discount mass in a proper way? (how much is enough)
- How to redistribute mass? (define neighbors)

Smoothing techniques for Markov Models

General Idea: Discount and redistribute

- Reserve mass from seen events in order to give to unseen events.

But... the devil is in the details!

- How to discount mass in a proper way? (how much is enough)
- How to redistribute mass? (define neighbors)
- How can we combine different model estimates and benefit from the complementary strengths of different models (Interpolation)?

Naive smoothing: Adding λ method (1)

Events: Set E of possible events, e.g. bigrams over V: $E = (V \times V)$

Data: $e_1, \ldots e_N$ (data size is N events)

Counting: Event (bigram) e occurs $C(e)$ times in Data

Relative Frequency estimate: $P_{rf}(e) = \frac{C(e)}{N}$

Naive smoothing: Adding λ method (1)

Events: Set E of possible events, e.g. bigrams over V: $E = (V \times V)$

Data: $e_1, \ldots e_N$ (data size is N events)

Counting: Event (bigram) e occurs $C(e)$ times in Data

Relative Frequency estimate: $P_{rf}(e) = \frac{C(e)}{N}$

Smoothing:

Add $0 < \lambda \leq 1$: for all $e \in E$ (bigrams) change $C(e)$ and $P_{rf}(e)$

$$\hat{C}(e) = C(e) + \lambda$$

$$P_\lambda(e) = \frac{C(e) + \lambda}{N + \lambda|E|}$$

Add λ method (2)

Example: Bigram Model

$$P_\lambda(w_i|w_{i-1}) = \frac{\lambda + c(w_{i-1}w_i)}{\sum_{w_i}(\lambda + C(w_{i-1}, w_i))}$$

Advantages: very simple and easy to apply

Disadvantages: Method performs poorly:

- All unseen events receive same probability!
- All events upgraded by λ!

Good-Turing method

Suppose we have data with total count of events being N:

Standard notation:

$$\begin{array}{lcl} r & = & \text{frequency of event } e \\ n_r & = & \text{number of events } e \text{ with frequency } r \\ & & n_r = |\{e \in E | Count(e) = r\}| \\ N_r & = & \text{total frequency of events occuring } r \text{ times} \\ & & N_r = r \times n_r \end{array}$$

Observation: $N = \sum_{r=1}^{\infty} N_r$ $N_0 = 0$

What we want: To recalculate the frequency r of an event (r^*)

Good-Turing Estimates

For events e with frequency r, the Good-Turing estimate r^* is given by:

$$r^* = (r+1) \times \frac{n_{r+1}}{n_r}$$

n_r :	number of events with freq. r
n_{r+1} :	number of events with freq. $r+1$

The Good-Turing probability estimate for events with frequency r is given by

$$P_{GT}(e) \approx \frac{r^*}{N}$$

Simple Good-Turing

For events e with frequency r, the Good-Turing estimate r^* is given by:

$$r^* = (r+1) \times \frac{n_{r+1}}{n_r}$$

If n_0 is unknown, estimate it using a linear regression on the frequency spectrum in logspace (Gale & Sampson, 1995).

r (MLE)	0	1	2	3	4
N_r	75×10^9	2×10^6	450000	189000	106000
c^* (GT)	0.0000270	0.446	1.26	2.24	3.24

Reference: Good-Turing Frequency Estimation without Tears, William A Gale and Geoffrey Sampson (1995).

Properties of Good-Turing

Preservation: Total number of counts is preserved:

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

Discounting: Total freq. for non-zero events is discounted

$$N_0 = n_0 \times 0^* = n_0 \times (1 \times \frac{n_1}{n_0}) = n_1$$

Zero freq. events

$$P_0 = \frac{r^*}{N} = \frac{0*}{N} = \frac{n_1}{N}$$

Zero events: No explicit method for redistributing N_0 among zero events!

Properties of Good-Turing

Preservation: Total number of counts is preserved:

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

Discounting: Total freq. for non-zero events is discounted

$$N_0 = n_0 \times 0^* = n_0 \times (1 \times \frac{n_1}{n_0}) = n_1$$

Zero freq. events

$$P_0 = \frac{r^*}{N} = \frac{0*}{N} = \frac{n_1}{N}$$

Zero events: No explicit method for redistributing N_0 among zero events!

GT provides a principled approach to discounting; do we now redistribute reserved mass (N_0) uniformly among zero events?

Why Uniform Redistribution is Not Good

Given some data, we observe that

- Frequency of trigram $\langle in, birds, of \rangle$ is zero

Why Uniform Redistribution is Not Good

Given some data, we observe that

- Frequency of trigram $\langle in, birds, of \rangle$ is zero
- Frequency of trigram $\langle power, hide, study \rangle$ is also zero

Why Uniform Redistribution is Not Good

Given some data, we observe that

- Frequency of trigram $\langle in, birds, of \rangle$ is zero
- Frequency of trigram $\langle power, hide, study \rangle$ is also zero
- Uniform distribution over unseen events means:

$$P(in, birds, of \mid in, birds) = P(power, hide, study \mid power, hide)$$

Why Uniform Redistribution is Not Good

Given some data, we observe that

- Frequency of trigram $\langle in, birds, of \rangle$ is zero
- Frequency of trigram $\langle power, hide, study \rangle$ is also zero
- Uniform distribution over unseen events means:

$$P(in, birds, of \mid in, birds) = P(power, hide, study \mid power, hide)$$

- Does that refelect our knowledge about English use?

$$P(in, birds, of \mid in, birds) > P(power, hide, study \mid power, hide)$$

Why Uniform Redistribution is Not Good

Given some data, we observe that

- Frequency of trigram $\langle in, birds, of \rangle$ is zero
- Frequency of trigram $\langle power, hide, study \rangle$ is also zero
- Uniform distribution over unseen events means:

$$P(in, birds, of \mid in, birds) = P(power, hide, study \mid power, hide)$$

- Does that refelect our knowledge about English use?

$$P(in, birds, of \mid in, birds) > P(power, hide, study \mid power, hide)$$

- This might be achieved if we look at bigrams!
  $P(in, birds \mid in)\ P(birds, of \mid birds)$
  vs.
  $P(power, hide \mid power)\ P(hide, study \mid hide)$

Recap

- Relation between language model and next word prediction
- Markov assumptions: time invariance & limited history
- Extracting ngrams from corpora
- Relation between ngrams, HMMs and PCFGs

Generalization & Zipf's law

- Learning $\approx$ Generalization to unseen data
- Zipf's law: the fat head and the long tail

Smoothing (I)

- Naive add-λ
- Good-Turing smoothing

Alan Turing

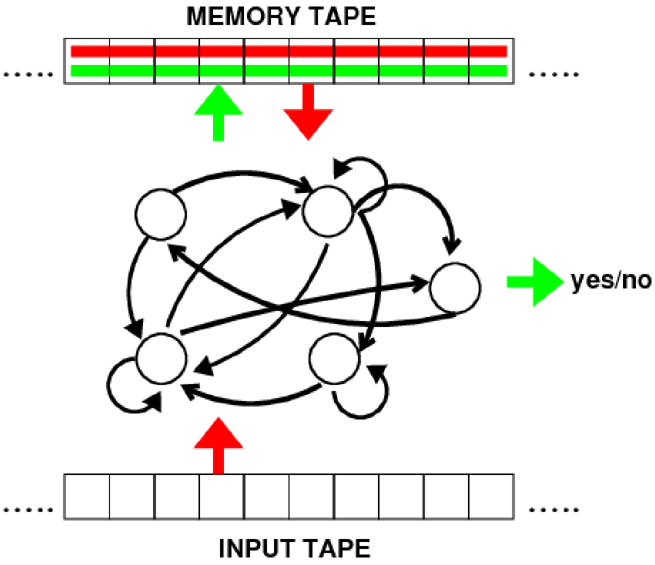
Smoothing (II)

- Katz back-off
- Jelinek-Mercer
- from ngrams to PCFGs

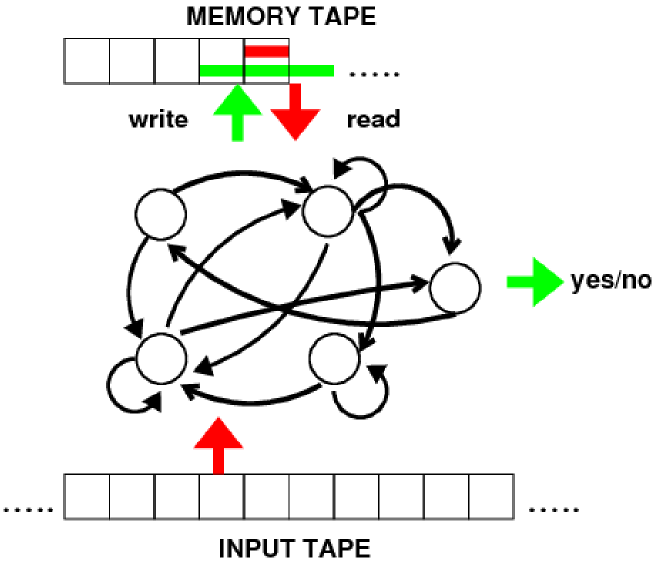
Alan Turing (1912-1954)

- Turing machine (1936)
- War-time decoding, Enigma
- Postwar: stored program computer (1945-46 Secret report on ACE; more complete than Von Neumann's EDVAC), involved in Manchester Mark I.
- Turing test (1950)
- Good-Turing (Good, 1953: Biometrika)
- Turing patterns - partial differential equations

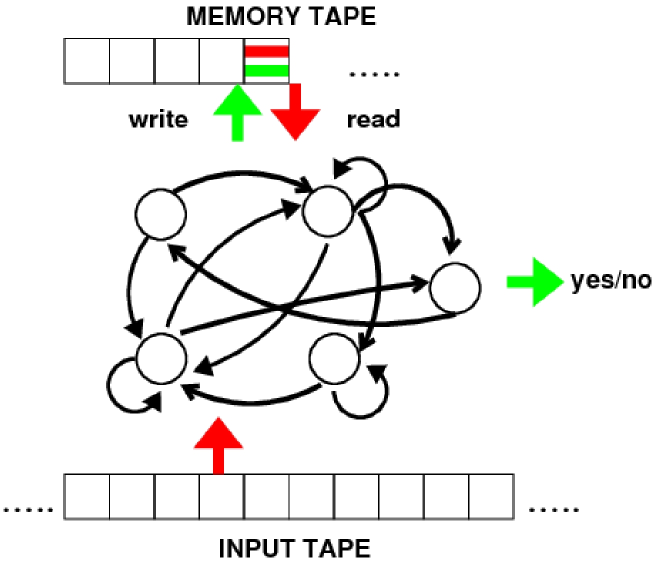
Turing Machine: Level 0 on the Chomsky Hierarchy



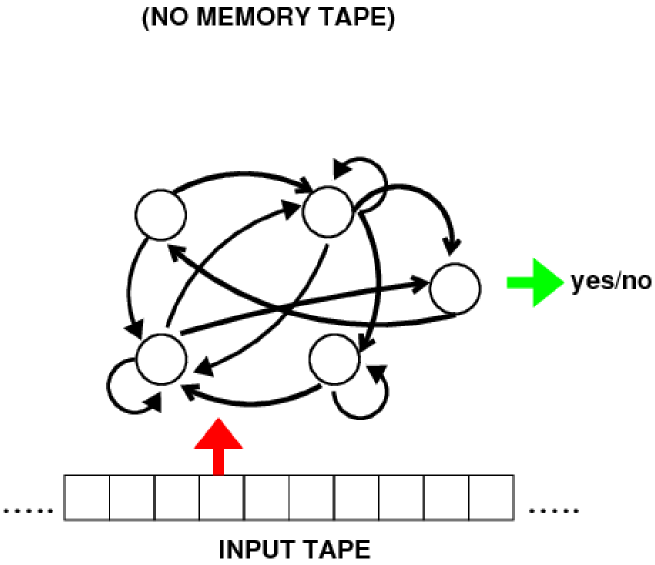
Embedded Pushdown Automaton: Level 1



Pushdown Automaton: Level 2 (Context-free)

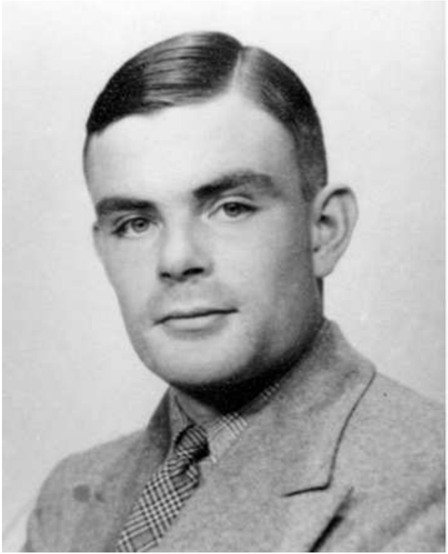


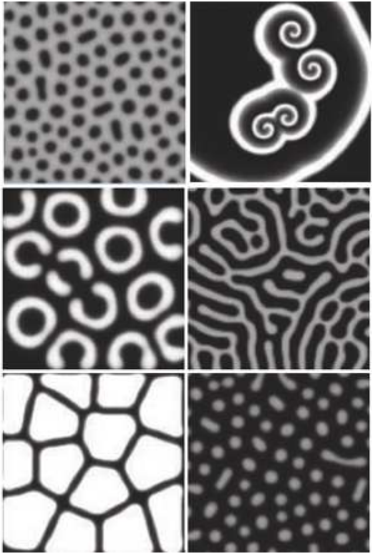
Finite-state Machine: Level 3 (Regular)



Enigma

Turing Patterns





Recap

- Relation between language model and next word prediction
- Markov assumptions: time invariance & limited history
- Extracting ngrams from corpora
- Relation between ngrams, HMMs and PCFGs

Generalization & Zipf's law

- Learning $\approx$ Generalization to unseen data
- Zipf's law: the fat head and the long tail

Smoothing (I)

- Naive add-λ
- Good-Turing smoothing

Alan Turing

Smoothing (II)

- Katz back-off
- Jelinek-Mercer
- from ngrams to PCFGs

Katz's Backoff

In order to re-distribute reserved mass, use a lower-order distribution

Discount and backoff Suppose $C(w_{i-1}w_i) = r$:

$$C_{katz}(w_{i-1}w_i) \;=\; \begin{cases} d_r \times r & \text{if } r > 0 \\ \alpha(w_{i-1}) \times P_{ML}(w_i) & \text{if } r = 0 \end{cases}$$

Discounting all non-zero counts are discounted by d_r ($\approx \frac{r^*}{r}$),

Redistribution of reserved mass over zero-events is proportional to their lower-order distribution, for bigrams this is unigrams, for trigrams this is bigrams etc.

Computing d_r

Efficiency: discount only events occuring $1 \leq r \leq k$: e.g. $k = 5$,

Constraint 1: Use Good-Turing estimate for discounting.
(There exists some constant μ such that
$d_r = 1 - \mu(1 - \frac{r^*}{r})$)

Constraint 2: What has been discounted must be redistributed:
Total mass for zero-events is equal to the discounted mass.
According to Good-Turing
$N_0 \ = \ n_0 0^* \ = \ n_0(\frac{n_1}{n_0}) \ = \ n_1$:

$$\sum_{r=1}^{k} n_r \times [r - (d_r r)] = n_1$$

Solution for d_r

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$$

Solution for $\alpha(w_{i-1})$ is through solving the constraint that total probability of all events (zero and non-zero occuring) must be equal to 1.0.

See extracts from Chen and Goodman Tech. Report. for details.

Smoothing of Markov Model

Smoothing an $(n+1)$ model gives us an n-gram model.

$$P^*(w_{n+1}|w_1,\ldots,w_n) = \frac{Count^*(w_1,\ldots,w_n,w_{n+1})}{\sum_{w\in V} Count^*(w_1,\ldots,w_n,w)}$$

Here,

$$\sum_{w\in V} Count^*(w_1,\ldots,w_n,w) \neq Count^*(w_1,\ldots,w_n)$$

Smoothing by linear interpolation: Jelinek-Mercer

(Also called finite mixture models)

Interpolate a bigram with a unigram model using some $0 \leq \lambda \leq 1$

$$P_{interp}(w_{i-1}w_i) = \lambda P_{ML}(w_{i-1}w_i) + (1 - \lambda)P_{ML}(w_i)$$

Estimation: data for P_{ML} and held-out data for estimating the λ's

Algorithm: Baum-Welch/Forward-Backward for Hidden Markov Models searching for a Maximum-Likelihood estimate of the λ.

More on this when we learn HMMs!

Ngrams: natural backoff hierarchy

... > Tetragrams > Trigrams > Bigrams > Unigrams > constant

Ngrams: natural backoff hierarchy

$$\ldots > \text{Tetragrams} > \text{Trigrams} > \text{Bigrams} > \text{Unigrams} > \text{constant}$$

$$P^*(z|wxy) = \lambda_4 P_{ML}(z|wxy) + (1-\lambda_4)P^*(z|xy)$$

Ngrams: natural backoff hierarchy

... > Tetragrams > Trigrams > Bigrams > Unigrams > constant

$$
\begin{aligned}
P^*(z|wxy) &= \lambda_4 P_{ML}(z|wxy) + (1-\lambda_4) P^*(z|xy) \\
P^*(z|xy) &= \lambda_3 P_{ML}(z|xy) + (1-\lambda_3) P^*(z|y)
\end{aligned}
$$

Ngrams: natural backoff hierarchy

... $>$ Tetragrams $>$ Trigrams $>$ Bigrams $>$ Unigrams $>$ constant

$$\begin{aligned} P^*(z|wxy) &= \lambda_4 P_{ML}(z|wxy) + (1-\lambda_4)P^*(z|xy) \\ P^*(z|xy) &= \lambda_3 P_{ML}(z|xy) + (1-\lambda_3)P^*(z|y) \\ P^*(z|y) &= \lambda_2 P_{ML}(z|y) + (1-\lambda_4)P^*(z) \end{aligned}$$

Ngrams: natural backoff hierarchy

... > Tetragrams > Trigrams > Bigrams > Unigrams > constant

$$
\begin{aligned}
P^*(z|wxy) &= \lambda_4 P_{ML}(z|wxy) + (1-\lambda_4)P^*(z|xy) \\
P^*(z|xy) &= \lambda_3 P_{ML}(z|xy) + (1-\lambda_3)P^*(z|y) \\
P^*(z|y) &= \lambda_2 P_{ML}(z|y) + (1-\lambda_4)P^*(z) \\
P^*(z) &= \lambda_1 P_{ML}(z) + (1-\lambda_4)c
\end{aligned}
$$

Ngrams: natural backoff hierarchy

$\ldots >$ Tetragrams $>$ Trigrams $>$ Bigrams $>$ Unigrams $>$ constant

$$
\begin{aligned}
P^*(z|wxy) &= \lambda_4 P_{ML}(z|wxy) + (1-\lambda_4)P^*(z|xy) \\
P^*(z|xy) &= \lambda_3 P_{ML}(z|xy) + (1-\lambda_3)P^*(z|y) \\
P^*(z|y) &= \lambda_2 P_{ML}(z|y) + (1-\lambda_4)P^*(z) \\
P^*(z) &= \lambda_1 P_{ML}(z) + (1-\lambda_4)c
\end{aligned}
$$

$$
\begin{aligned}
P^*(z|wxy) &= \lambda_{wxyz} P_{ML}(z|wxy) + (1-\lambda_{wxyz})P^*(z|xy) \\
&\cdots
\end{aligned}
$$

Ngrams: natural backoff hierarchy

... > Tetragrams > Trigrams > Bigrams > Unigrams > constant

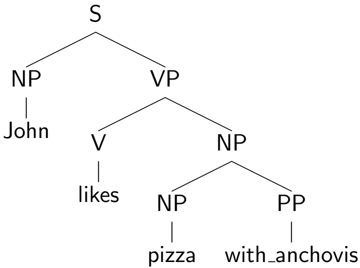
$$
\begin{aligned}
P^*(z|wxy) &= \lambda_4 P_{ML}(z|wxy) + (1-\lambda_4)P^*(z|xy) \\
P^*(z|xy) &= \lambda_3 P_{ML}(z|xy) + (1-\lambda_3)P^*(z|y) \\
P^*(z|y) &= \lambda_2 P_{ML}(z|y) + (1-\lambda_4)P^*(z) \\
P^*(z) &= \lambda_1 P_{ML}(z) + (1-\lambda_4)c
\end{aligned}
$$

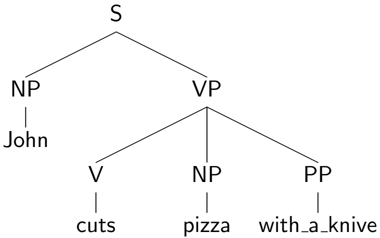
$$
\begin{aligned}
P^*(z|wxy) &= \lambda_{wxyz} P_{ML}(z|wxy) + (1-\lambda_{wxyz})P^*(z|xy) \\
&\ldots
\end{aligned}
$$

Deleted interpolation: for each next level, we delete the conditioning context *furthest back in the past*.

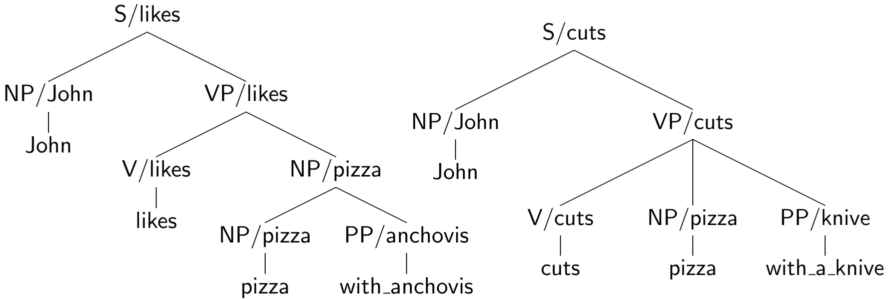
Need for smoothing in PCFGs

Head-lexicalisation in PCFGs

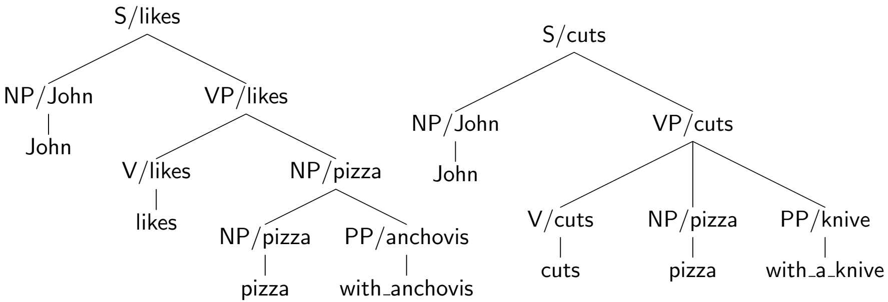




Head-lexicalisation in PCFGs

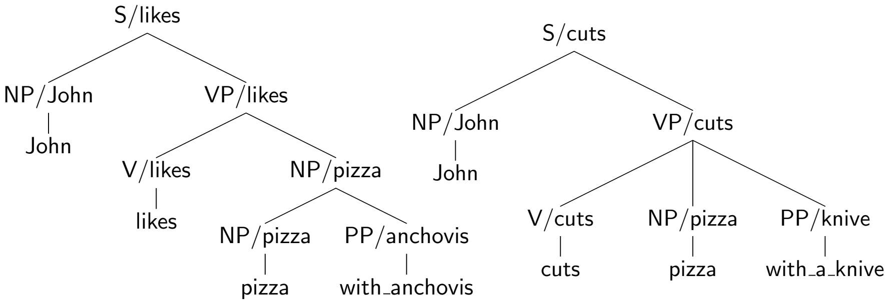


Head-lexicalisation in PCFGs



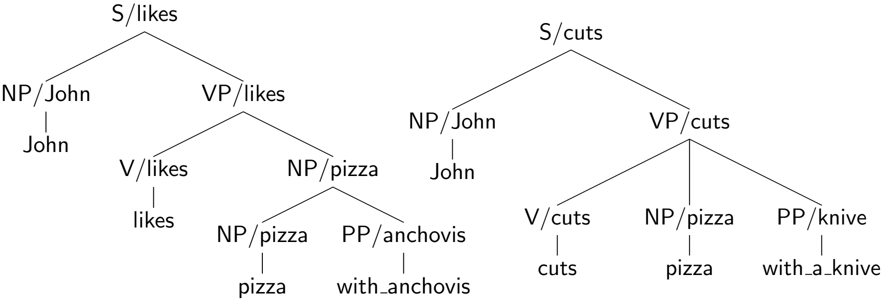
VP/likes $\rightarrow$ V/likes NP/pizza	.3
VP/likes $\rightarrow$ V/likes NP/pasta	.7
VP/cuts $\rightarrow$ V/cuts NP/pizza NP/knive	.6
VP/cuts $\rightarrow$ V/cuts NP/pasta NP/knive	.4
NP/pizza $\rightarrow$ NP/pizza PP/anchovis	.1
NP/pizza $\rightarrow$ pizza	.9

Head-lexicalisation in PCFGs



John cuts pizza with anchovis
John cuts bread with a knive

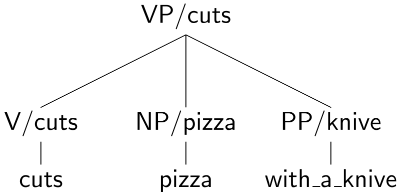
Head-lexicalisation in PCFGs



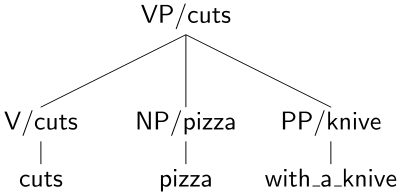
John cuts pizza with anchovis
John cuts bread with a knive

Need to smooth lexicalized rules!

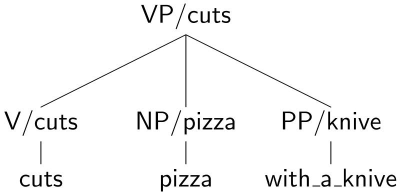
Back-off levels in PCFGs

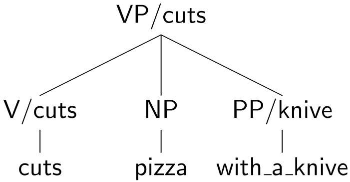


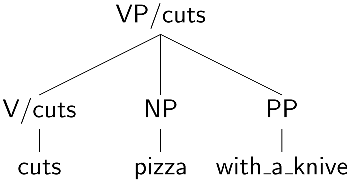
Back-off levels in PCFGs



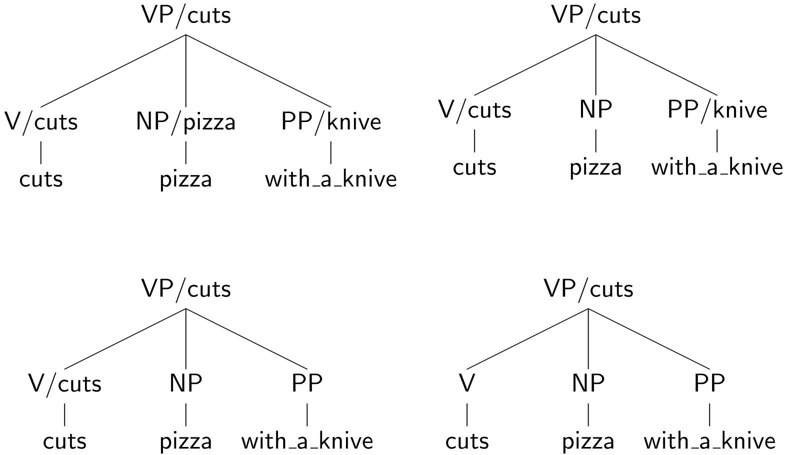
Back-off levels in PCFGs







Back-off levels in PCFGs



Deleted interpolation

$$P(\text{expansions}|\text{parent, grandparent, leftsister, lexical head}) = ?$$

Deleted interpolation

$$P(\text{expansions}|\text{parent, grandparent, leftsister, lexical head}) = ?$$

Priority list

1. parent
2. lexical head
3. grandparent
4. left sister

Deleted interpolation

$$P(\text{expansions}|\text{parent, grandparent, leftsister, lexical head}) = ?$$

Priority list

1. parent
2. lexical head
3. grandparent
4. left sister

$$P(\text{expansions}|\text{parent, grandparent, leftsister, lexical head}) =$$

$\lambda_0 P_{MLE}(\ldots) +$
$(1\text{-}\lambda_0) P(\text{expansions}|\text{parent, grandparent, lexical head})$
$\ldots$

No natural hierarchy in PCFG

Deleted interpolation: for each next level, we delete the conditioning context according to our best guess of the least relevant information.

Summary

- n-gram statistics suffer from sparseness in text as n grows
- There are smoothing methods against sparseness
- Add λ is too simple
- Good-Turing only for reserving mass for zero-events
- Katz allows redistribution according to a lower order distribution
- Interpolation combines lower order distributions

Reference: Joshua Goodman and Stanly Chen. *An empirical study of smoothing techniques for language modeling*. Tech. report TR-10-98, Harvard University, August 1998.
`http://research.microsoft.com/~joshuago/`

Next Class

Hidden Markov Models

- Theory of Hidden Markov Models
- POS tagging: Implementing the language model and the lexical model
- Smoothing
- Efficient POS taggers: Trellis, forward/backward algorithm.

No class next week on Monday (Easter)