

Today's Topic: Language Modeling Part 1

Reference: Review paper from Roni Rosenfeld

(paper online on our course webpage)

Xuedong Huang et al. Spoken Language Processing, Chapter 11

Language Modeling (Part 1)

- Motivation and Definition
- Language Modeling in ASR
- Perplexity
- Parameter estimation
- Smoothing techniques

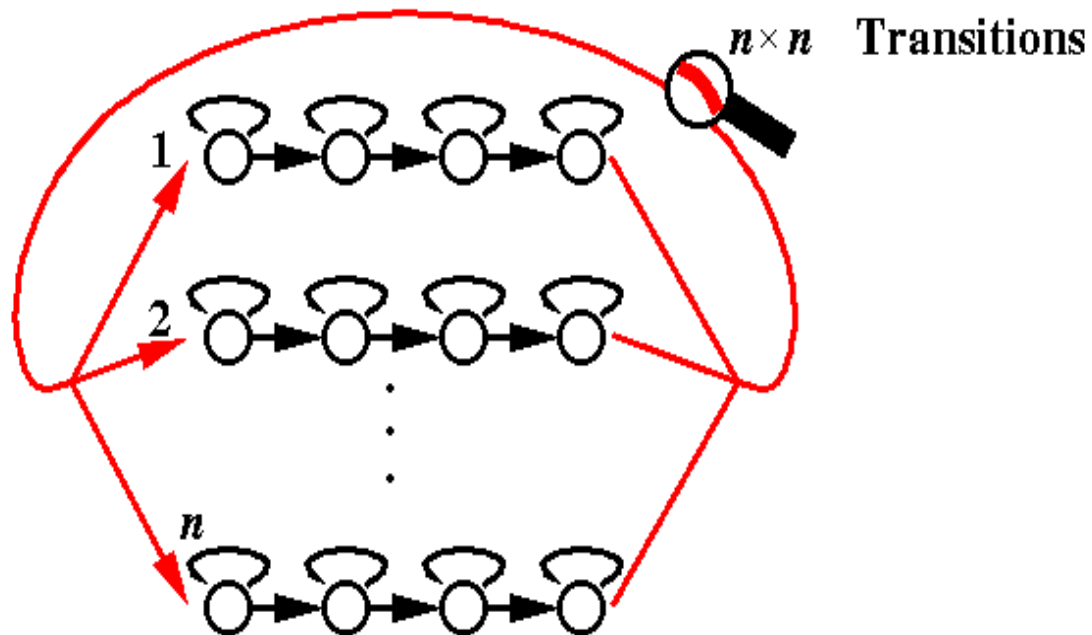
Stochastic Language Models

- In formal language theory $P(W)$ is regarded either as
 - 1.0 if word sequence W is accepted
 - 0.0 if word sequence W is rejected
- Inappropriate for spoken language since,
 - grammar has no complete coverage
 - (conversational) spoken language is often ungrammatical
- Describe $P(W)$ from the probabilistic viewpoint
 - Occurrence of word sequence W is described by a probability $P(W)$
 - find a good way to accurately estimate $P(W)$
- Training problem: reliably estimate probabilities of W
- Recognition problem: compute probabilities for generating W

Deterministic vs. Stochastic Language Models

In HMM-recognizers, the language model is responsible for the computation of the word-to-word transition probabilities.

These can be computed on the fly, and may depend on more than just the previous word.



LMs can be

deterministic: $P(w_j|w_i) = 0.0$ or $1.0/n$ (e.g. finite state grammars)

stochastic: transition probabilities are in the range 0.0 to 1.0

A Word Guessing Game

Good morning, how are _____

A Word Guessing Game

Good morning, how are **you**?

I apologize for being late, I am very _____

A Word Guessing Game

Good morning, how are you?

I apologize for being late, I am very **sorry!**

A Word Guessing Game

Good morning, how are you?

I apologize for being late, I am very sorry!

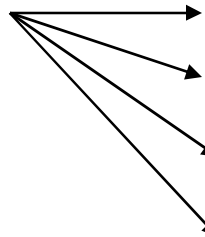
My favorite OS is _____

A Word Guessing Game

Good morning, how are you?

I apologize for being late, I am very sorry!

My favorite OS is



Linux

Unix

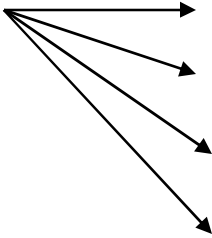
Windows XP

WinCE

A Word Guessing Game

Good morning, how are you?

I apologize for being late, I am very sorry!

My favorite OS is  **Linux**
Unix
Windows XP
WinCE

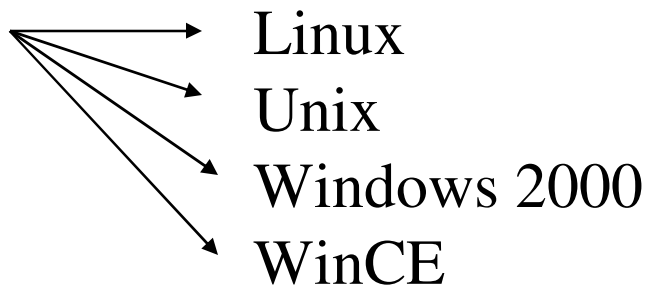
Hello, I am very happy to see you, Mr. _____

A Word Guessing Game

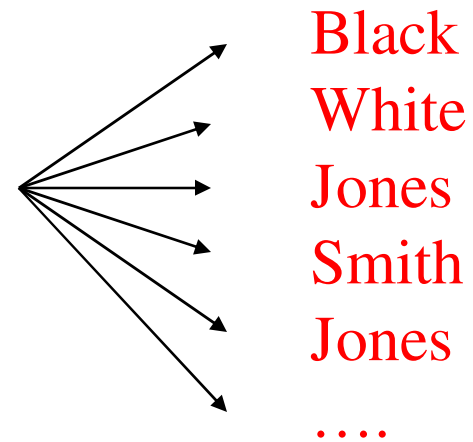
Good morning, how are you?

I apologize for being late, I am very sorry!

My favorite OS is



Hello, I am very happy to see you, Mr.



A Word Guessing Game

What do we learn from the word guessing game?

- for some histories the number of expected words is rather small.
- for some histories we can make virtually no prediction about the next word.
- the more words fit at some point the more difficult it is to recognize the correct one (more errors are possible)
- the difficulty of recognizing a word sequence is correlated with the "branching degree"

What do we expect from Language Models in SR?

- **Improve speech recognizer**

add another information source

- **Disambiguate homophones**

find out that "I OWE YOU TOO" is more likely
than "EYE O U TWO"

- **Search space reduction**

when vocabulary is n words, don't consider all n^k
possible k -word sequences

- **Analysis**

analyze utterance to *understand* what has been said
disambiguate homonyms (bank: money vs river)

Language Modeling in Automatic Speech Recognition

Remember the fundamental problem of speech recognition:

Given: An Observation $X = x_1, x_2, \dots, x_T$

Wanted: $W' = w'_1, w'_2, \dots, w'_n$ with highest likelihood:

$$W' = \operatorname{argmax}_W P(W | X) = \frac{p(X | W) \cdot P(W)}{p(X)} = \operatorname{argmax}_W p(X | W) \cdot P(W)$$

This poses four problems to the speech recognizer:

- What is X ? The problem of preprocessing.
- What is $p(X | W)$. The acoustic modeling.
- What is $P(W)$. The language modeling.
- How do we find the argmax_W ? The search problem.

Probabilities of Word Sequences

The probability of a word sequence can be decomposed as:

$$P(W) = P(w_1 w_2 \dots w_n) = P(w_1) \cdot P(w_2 | w_1) \cdot P(w_3 | w_1 w_2) \cdot \dots \cdot P(w_n | w_1 w_2 \dots w_{n-1})$$

The choice of w_n thus depends on the entire history of the input, so when computing $P(w | \text{history})$, we have a problem:

For a vocabulary of 64,000 words and average sentence lengths of 25 words (typical for Wall Street Journal), we end up with a huge number of possible histories ($64,000^{25} > 10^{120}$).

So it is impossible to precompute a special $P(w | \text{history})$ for every history.

Two possible solutions:

- compute $P(w | \text{history})$ "on the fly" (rarely used, very expensive)
- replace the history by one out of a limited feasible number of equivalence classes C such that $P'(w | \text{history}) = P(w | C(\text{history}))$

Question: how do we find good equivalence classes C ?

Classification of Word Sequence Histories

We can use different equivalence classes using information about:

- Grammatical content (phrases like noun-phrase, etc.)
- POS = part of speech of previous word(s) (e.g. subject, object, ...)
- Semantic meaning of previous word(s)
- Context similarity (words that are observed in similar contexts are treated equally, e.g. weekdays, people's names etc.)
- Apply some kind of automatic clustering (top-down, bottom-up)
- Classes are simply based on previous words
 - **unigram:** $P'(w_k | w_1 w_2 \dots w_{k-1}) = P(w_k)$
 - **bigram:** $P'(w_k | w_1 w_2 \dots w_{k-1}) = P(w_k | w_{k-1})$
 - **trigram:** $P'(w_k | w_1 w_2 \dots w_{k-1}) = P(w_k | w_{k-2} w_{k-1})$
 - ***n*-gram:** $P'(w_k | w_1 w_2 \dots w_{k-1}) = P(w_k | w_{k-(n-1)} w_{k-n-2} \dots w_{k-1})$

Estimation of N-grams

The standard approach to estimate $P(w|history)$ is

- use a large amount of training corpus
(There's no data like more data)
- determine the FREQUENCY with which the word w occurs given the *history*
- count how often the word sequence “*history w*” occurs in the text
- normalize the count by the number of times *history* occurs

$$P(w|history) = \frac{\text{Count}(\text{history } w)}{\text{Count}(\text{history})}$$

Example

Training corpus consists of 3 sentences, use bigram model

John read her book.

I read a different book.

John read a book by Mulan.

$$P(w| \text{history}) =$$

$$\frac{\text{Count}(\text{history } w)}{\text{Count}(\text{history})}$$

$$\text{Count}(\text{history})$$

$$P(\text{John} | \langle s \rangle) = C(\langle s \rangle, \text{John}) / C(\langle s \rangle) = 2/3$$

$$P(\text{read} | \text{John}) = C(\text{John}, \text{read}) / C(\text{John}) = 2/2$$

$$P(\text{a} | \text{read}) = C(\text{read}, \text{a}) / C(\text{read}) = 2/3$$

$$P(\text{book} | \text{a}) = C(\text{a}, \text{book}) / C(\text{a}) = 1/2$$

$$P(\langle /s \rangle | \text{book}) = C(\text{book}, \langle /s \rangle) / C(\text{book}) = 2/3$$

Calculate the probability of sentence *John read a book.*

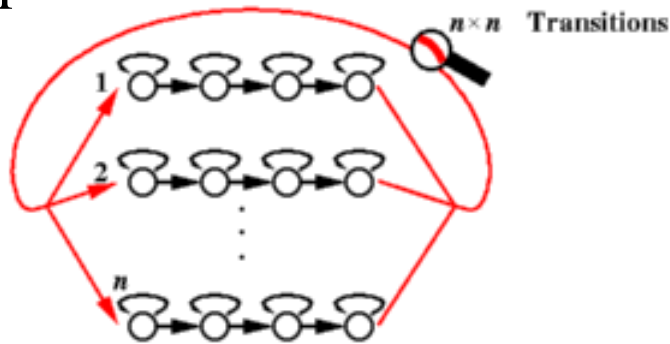
$$P(\text{John read a book}) = P(\text{John} | \langle s \rangle) P(\text{read} | \text{John}) P(\text{a} | \text{read}) P(\text{book} | \text{a}) P(\langle /s \rangle | \text{book}) = 0.148$$

What about the sentence *Mulan read her book.*

We don't have $P(\text{read} | \text{Mulan})$

Bigrams vs. Trigrams

Bigrams can be easily incorporated into an HMM recognizer:



For trigrams, we need a larger history. What if a word can have many predecessors?

Typical solution for incorporating trigrams:

- use time asynchronous search (easier to handle long history)
- for time-synchronous search: use "poor man's trigrams", i.e. consider only the predecessor's *best* predecessor instead of all.

Other disadvantages of trigrams compared to bigrams:

- coverage of test data is smaller than with bigrams
- estimation of $P(w_k | w_{k-2} w_{k-1})$ is more difficult

Typical error reductions: **bigrams** 30%-50%, **trigrams** 10%-20%, **fourgrams** 3%-5%

Measuring the Quality of Language Models

Obvious approach to finding out whether LM1 or LM2 is better:

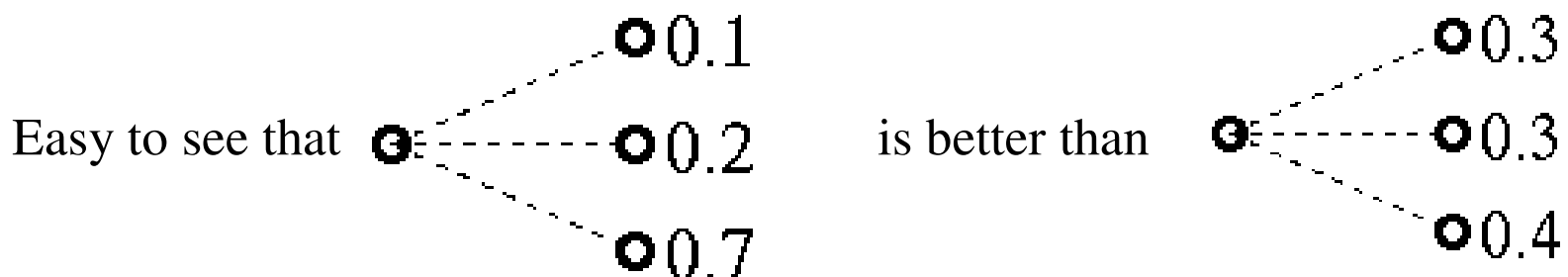
Let recognizer run with both and choose the one that produces fewer errors.

But:

- Performance of recognizer depends also on acoustic model.
- Performance of recognizer depends also on combination mechanism of acoustic model with language model.
- Expensive and time consuming
- What if no recognizer available

We would like to have an independent measure:

Declare a LM to be good, if it makes the task easier for the recognizer (i.e. if it has a smaller average "branching factor").



The Perplexity of a Language Model

Language can be thought of as an information source whose output are words w_i belonging to the vocabulary of that language

The entropy of an information source emitting words $w_1 w_2 \dots$ is defined as:

$$H(W) = - \lim_{n \rightarrow \infty} (1/n) \sum_{w_1 w_2 \dots w_n} P(w_1 w_2 \dots w_n) \cdot \log P(w_1 w_2 \dots w_n)$$

Equivalently, a source of entropy H is one that has as much information content as a source that puts out words equiprobably from a vocabulary of 2^H words.

For ergodic sources (can be characterized by "infinite" word sequence) the entropy can be computed as:

$$H(W) = - \lim_{n \rightarrow \infty} (1/n) \log P(w_1 w_2 \dots w_n)$$

Since we never have an infinite number of words, we approximate H by using a very large text corpus, with a very large n (e.g. hundreds of millions words). Then we simply approximate:

$$H(W) = (1/n) \log P(w_1 w_2 \dots w_n)$$

The Perplexity of a Language Model

The entropy of a source $H(W) = (1/n) \log P(w_1 w_2 \dots w_n)$ is a measure for the difficulty for a recognizer to recognize the source's language.

High entropy -> difficult task.

Low entropy -> easy task.

If we consider P as *the* language model, then the entropy is also a measure for the quality of a language model.

The term:

$$PP = 2^{H(W)} = P(w_1 w_2 \dots w_n)^{-1/n}$$

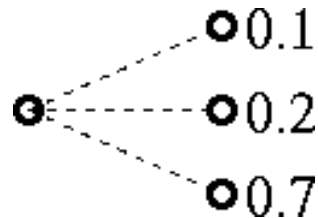
Is called the **perplexity** of a language model P on the test set $W = w_1 w_2 \dots w_n$.
The perplexity can also be regarded as the geometric mean of all branching factors.
Recognizing a language of perplexity PP is equivalently difficult as recognizing a language that equiprobably emits words from a vocabulary of PP words.

The Perplexity of a Language Model

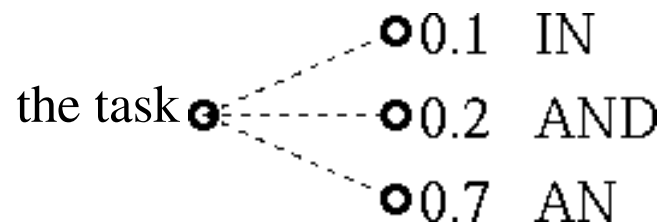
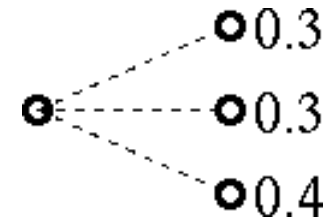
The perplexities of some tasks:

Task	Vocabulary	Language Model	Perplexity
Conference Registration	400	Bigrams	7
Resource Management	1000	Bigrams	20
Wall Street Journal	60.000	Bigrams	160
Wall Street Journal	60.000	Trigrams	130
Arabic Broadcast News	220.000	Fourgrams	212
Chinese Broadcast News	90.000	Fourgrams	430

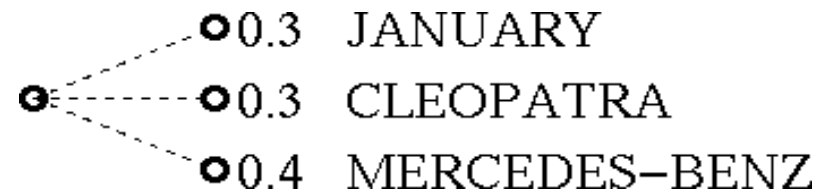
Problem: Even if the perplexity of



is lower than the one of



is more difficult than



Smoothing

Key problem in n-gram modeling is the **data sparseness** problem

- many possible word successions may not be well observed
- or even not be seen at all!
- Example: Given several million word collection of English text
 - 50% of trigrams occur only once
 - 80% of trigrams occur less than 5 times

↑ Smoothing is critical to make probabilities robust for unseen data

Example from last time: *Mulan read a book*

We had: $\text{Count}(Mulan, read)=0$, $P(read|Mulan) = \frac{\text{Count}(Mulan, read)}{\sum_w \text{Count}(Mulan, w)} = \frac{0}{1}$

In ASR if $P(W)=0$, the string W will never be considered as hypothesis, thus whenever a string W with $P(W)$ occurs, an error will be made

↑ Assign all strings a nonzero probability to prevent ASR errors

Smoothing concept

Steps:

- Subtract counts from **seen** events
- Redistribute collected counts to **unseen** events (count=0)

Analogy:

- Taxation for the rich and poor
- Collect tax from the rich
- Redistribute it to the poor

Smoothing strategies

1. Backoff smoothing
 - Absolute discounting
 - Katz
 - Kneser-Ney
2. Linear interpolation
 - Deleted interpolation

Backoff smoothing

Principle:

- Use **only** higher-order model whenever possible.
- Backoff if it is not reliable (unseen events)

Recursive definition:

$$P_{bo}(w_i|w_{i-1}) = \begin{cases} \hat{f}(w_i|w_{i-1}) & \text{if } C(w_{i-1}w_i) > 0 \\ \lambda(w_{i-1}) \cdot P_{bo}(w_i) & \text{otherwise} \end{cases}$$

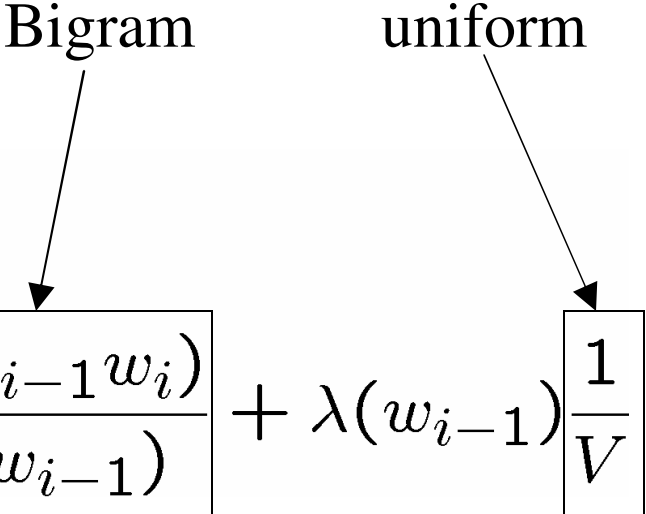
Question: How can we get the lambdas?

“Add-one” smoothing

Simple method: Add 1 to all bigram

$$\begin{aligned} P_{one}(w_i|w_{i-1}) &= \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V} \\ &= (1 - \lambda(w_{i-1})) \boxed{\frac{C(w_{i-1}w_i)}{C(w_{i-1})}} + \lambda(w_{i-1}) \boxed{\frac{1}{V}} \end{aligned}$$

Bigram uniform



⇒ **Not good** (What happen when V gets large?)
(in the homework)

Absolute discounting

Subtract a fixed D from each N -gram count:

$$P_{abs}(w_i|w_{i-1}) = \begin{cases} \frac{\max\{C(w_{i-1}w_i) - D, 0\}}{C(w_{i-1})} & \text{if } C(w_{i-1}w_i) > 0 \\ \lambda(w_{i-1}) \cdot P_{abs}(w_i) & \text{otherwise} \end{cases}$$

Good Turing Discounting

Pretend an event occurred r times to be r^*

n_r : # of events occurred r times

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

Discount: $d_r \approx \frac{r^*}{r}$

Katz Smoothing

- Event: a word bigram
- r : bigram count
- k : threshold to apply discounting
- Use Good-Turing for discounting

$$P_{katz}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i)}{C(w_{i-1})} & \text{if } r > k \\ d_r \cdot \frac{C(w_{i-1}w_i)}{C(w_{i-1})} & \text{if } k \geq r > 0 \\ \alpha(w_{i-1}) \cdot P_{katz}(w_i) & \text{if } r = 0 \end{cases}$$

Kneser-Ney smoothing

Motivation: think about $P(\text{Francisco} \mid \text{MU})$ instead of $P(\text{Francisco} \mid \text{San})$

- Backoff occur, but $P(\text{Francisco})$ is high \Rightarrow not intuitive
- Different strategy to model lower-order distribution
 - Use # of **unique preceding** words
 - e.g. $C(\text{Francisco}) = \#(* \text{Francisco}) = 1$
- Use absolute discounting, but with special backoff distribution:

$$P_{KN}(w_i) = \frac{\#(*w_i)}{\sum_v \#(*v)}$$

Linear interpolation

Different from backoff strategy:

- **All** lower-order distributions are interpolated

Recursive definition:

$$P_I(w_i|w_{i-1}w_{i-2}) = \lambda \cdot \hat{P}(w_i|w_{i-1}w_{i-2}) + (1 - \lambda) \cdot P_I(w_i|w_{i-1})$$

Deleted Interpolation Smoothing

Motivation

- $\text{Count}(\textit{like you}) = 0$, $\text{Count}(\textit{like Tarantula}) = 0$
- simple additive smoothing ($\text{Count}+1$ if $\text{Count}=0$): $P(\textit{you}|\textit{like}) = P(\textit{Tarantula}|\textit{like})$
- intuitively we rather prefer $P(\textit{you}|\textit{like}) > P(\textit{Tarantula}|\textit{like})$ since *you* is more common in English text than *Tarantula*
- Solution: Interpolate the bigram model with a unigram model

$$P_I(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda) P(w_i), \text{ with } 0 \leq \lambda \leq 1$$

- since $P(\textit{you}) > P(\textit{Tarantula})$, we get $P_I(\textit{you}|\textit{like}) > P_I(\textit{Tarantula}|\textit{like})$
- Elegant: recursively defined linear interpolation between n- and (n-1)th order LM

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

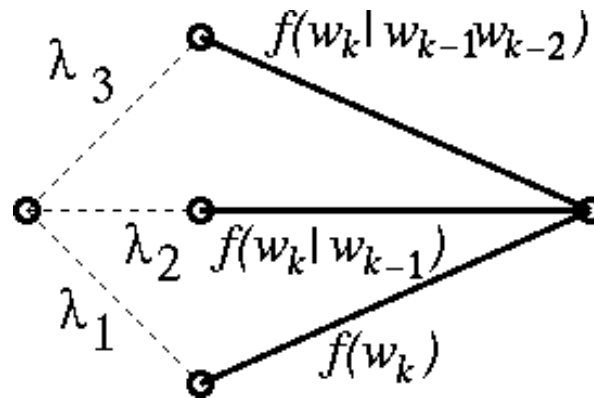
- End of the recursion could be the smoothed first-order model (unigram) or smoothed zero-order model to be the uniform distribution
- Notice $\lambda_{w_{i-n+1} \dots w_{i-1}}$ is different for different histories $w_{i-n+1} \dots w_{i-1}$
- i.e. for contexts we have seen a lot, a high λ is appropriate
- Training $\lambda_{w_{i-n+1} \dots w_{i-1}}$ independently requires lots of data, solution: tying of λ

Interpolation Factors for Parameter Smoothing

$$P(w_3 | w_1 w_2) = \lambda_1 \cdot \frac{f(w_1 w_2 w_3)}{f(w_1 w_2)} + \lambda_2 \cdot \frac{f(w_1 w_2)}{f(w_1)} + \lambda_3 \cdot \frac{f(w_1)}{\sum f(w_i)}$$

Obvious: To stay within probabilities (0.0..1.0), we must demand that $\sum_i \lambda_i = 1.0$

Some wishful thinking:
(**mixture model**)



Let $C(i)$ be the number of times the process visited state i . Then

$$\lambda_i = \frac{C(i)}{C(1) + C(2) + C(3)}$$

Interpolation Factors for Parameter Smoothing

Iterative Solution:

1. $\lambda_1^0 = \lambda_2^0 = \lambda_3^0 = 1/3$

2. $Q_k^j(1) = \lambda_1^j f(w_k | w_{k-1} w_{k-2})$

$Q_k^j(2) = \lambda_2^j f(w_k | w_{k-1})$

$Q_k^j(3) = \lambda_3^j f(w_k)$

3. $P_k^j(i) = Q_k^j(i) / (Q_k^j(1) + Q_k^j(2) + Q_k^j(3))$

4. let $\lambda_i^{j+1} = \frac{\sum_k P_k^j(i)}{\sum_k (P_k^j(1) + P_k^j(2) + P_k^j(3))}$

5. $j++$; goto step 2

The iterative solution corresponds to an iterative averaging of the λ s for all words w_k . Other way of finding good values for the λ s is deleted interpolation smoothing.

Performance of N-gram Smoothing

- Kneser-Ney outperforms other methods over a wide range of training set sizes and corpora for both 2- and 3-grams
- Next are Katz (many data) and deleted interpolation (sparse data)
- These smoothing algorithms are better than n-gram w/o smoothing
- Smoothing is necessary wherever data sparseness is an issue
- In case of enough training data to reliably calculate probabilities
 ↑ move to higher-order n-gram model ↑ data sparseness issue
- Table: 60k vocabulary, 250Mio Wall Street Journal, Whisper [XD]

<i>Models</i>	<i>Perplexity</i>	<i>WER</i>
Unigram Katz	1196,45	14,85
Unigram Kneser-Ney	1199,59	14,86
Bigram Katz	176,31	11,38
Bigram Kneser-Ney	176,11	11,34
Trigram Katz	95,19	9,69
Trigram Kneser-Ney	91,47	9,6

Class N-grams

- Another way to handle data sparseness: define classes of similar semantic or grammatical behavior
 - Rapid adaptation
 - training on small data sets
 - real-time systems (reduced memory requirements)
- For simplicity assume that one word w_i can be uniquely assigned to only one class c_i , then:
- $P(w_i | c_{i-n+1} \dots c_{i-1}) = P(w_i | c_i) P(c_i | c_{i-n+1} \dots c_{i-1})$
- Learn the mapping $w \leftrightarrow c$ from training text or task knowledge
- For general-purpose LVCSR applications class-based n-grams have NOT significantly improved the recognition accuracy
- In LVCSR rather used as backoff model for smoothing
- For limited-domain SR class-based n-grams are helpful since classes can efficiently encode semantic information, improve SU

Summary (Part 1)

- N-gram LM
- Application: Constrain search space in ASR
- Smoothing strategy
 - backoff
 - Katz (Good-Turing based), Absolute discounting, Kneser-Ney
 - linear interpolation
 - Deleted interpolation
- Class-based LM: Useful for semantic knowledge integration