T-61.184
Automatic Speech Recognition: From Theory to Practice

http://www.cis.hut.fi/Opinnot/T-61.184/
October 18, 2004

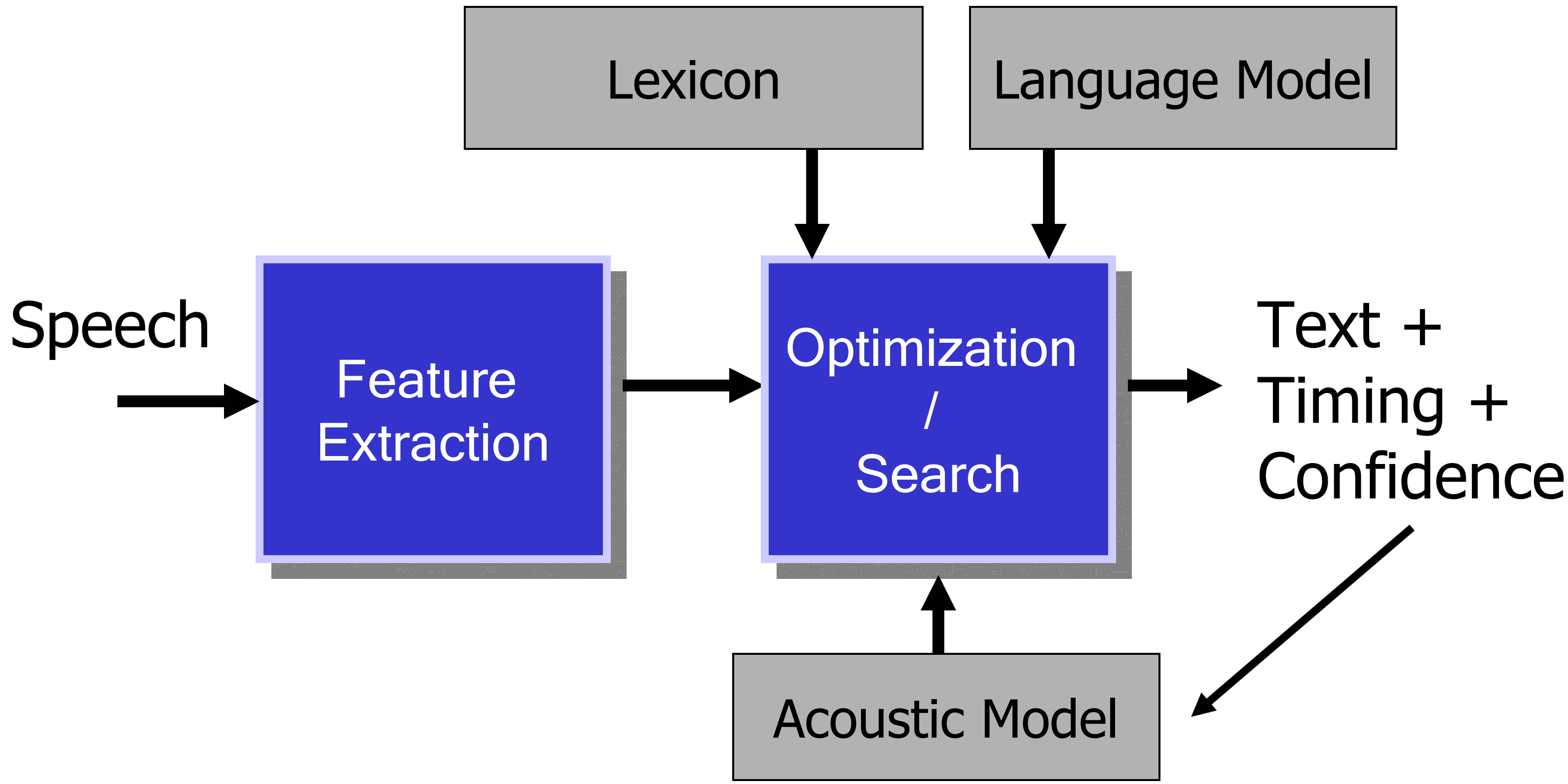
Prof. Bryan Pellom
Department of Computer Science
Center for Spoken Language Research
University of Colorado

pellom@cslr.colorado.edu

References for Today's Material

- S. M. Katz, "Estimation of Probabilities from Sparse Data for a Language Model Component of a Speech Recognizer," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. 35, No. 3, pp. 400-401, 1987.
- R. Kneser, H. Ney, "Improved Backing-Off for M-gram Language Modeling," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 1995.*
- X. Huang, A. Acero, H. Hon, Spoken Language Processing, *Prentice Hall, 2001*
- Joshua Goodman & Eugene Charniak, AAAI 2002 Language Modeling Tutorial

Speech Recognizer Block Diagram



Recall the Bayes Rule Formulation

- **Using Bayes Rule,**

$$P(\mathrm{W} \mid \mathrm{O}) = \frac{P(\mathrm{O} \mid \mathrm{W})P(\mathrm{W})}{P(\mathrm{O})}$$

- **Since P(O) does not impact optimization,**

$$\begin{aligned}\hat{\mathrm{W}} &= \underset{\mathrm{W}}{\arg\max}\, P(\mathrm{W} \mid \mathrm{O}) \\ &= \underset{\mathrm{W}}{\arg\max}\, P(\mathrm{O} \mid \mathrm{W})P(\mathrm{W})\end{aligned}$$

Practical Speech Recognition

- **In practice, we work with log-probabilities,**

$$\hat{W} = \arg\max_{W} \left\{ \log\left(P(O \mid W) P(W) \right) \right\}$$

- **Common to scale LM probabilities by a <u>grammar scale factor</u> ("s") and also include a <u>word-transition penalty</u> ("p"):**

$$\hat{W} = \arg\max_{W} \left\{ \underbrace{\log\left(P(O \mid W) \right)}_{\text{acoustic model}} + \underbrace{s \cdot \log\left(P(W) \right) + p}_{\text{language model}} \right\}$$

Language Models

- **Assign probabilities to word sequences** $P(\mathrm{W})$
- **Aids in reducing search space and ambiguity**
- **Resolves most homonyms:**

`Write a letter to Mr. Wright right away`

- **Constraint / Flexibility tradeoff**

Grammar-Based Language Models

- Encode valid word sequences as a finite state network
- Probabilities can be assigned to network nodes
- Only word sequences modeled by grammar can be spoken
- Requires human-designed grammar, no training data

Regular Expression Grammar

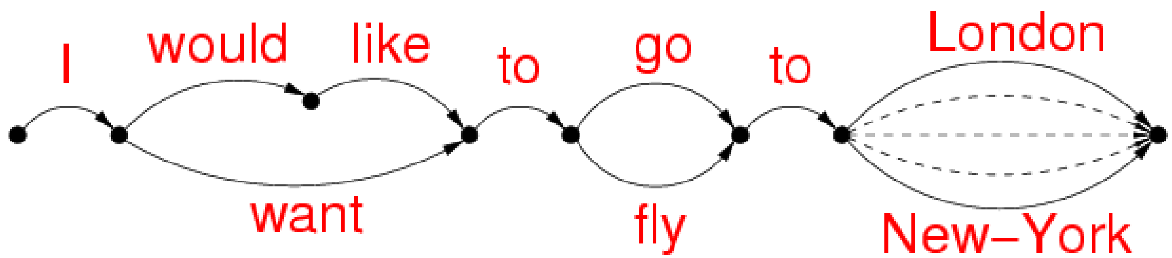
Grammar:

$$\langle\text{sentence}_1\rangle = \text{I} \left\{ \begin{array}{c} \text{would like} \\ \text{want} \end{array} \right\} \text{to} \left\{ \begin{array}{c} \text{go} \\ \text{fly} \end{array} \right\} \text{to} \ \langle\text{airport}\rangle$$

$$\langle\text{sentence}_2\rangle = \cdots$$

$$\langle\text{airport}\rangle = \{ \text{London}, \ \text{New–York}, \ldots \}$$

Finite–state representation:



Regular Expression Grammar

Example:

Cambridge HTK Recognizer Grammar format:

| alternatives

[ ] optional expressions

{ } zero or more repetitions

< > one or more repetitions

( ) defines task grammar

Regular Expression Grammar Example

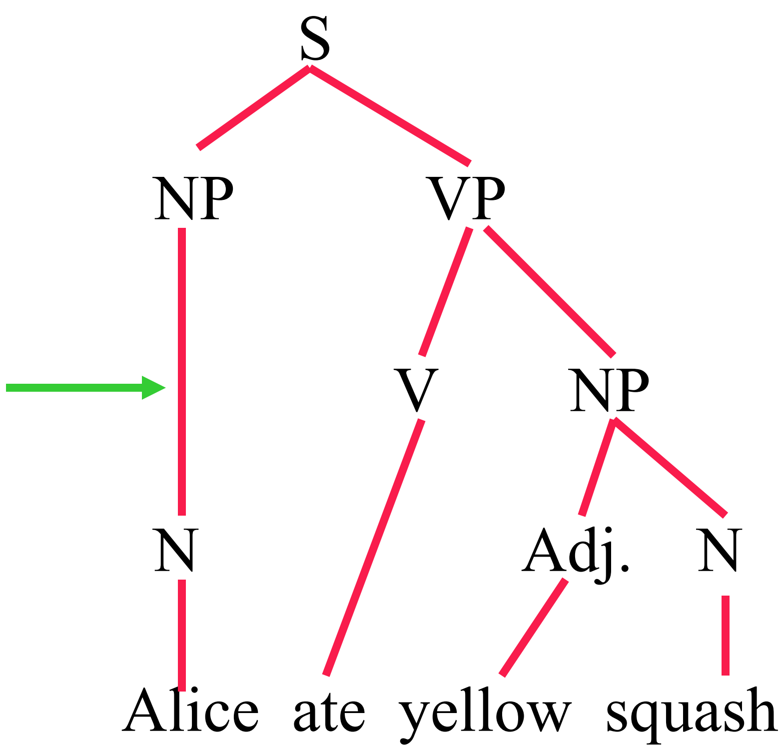
```
$digit = one|two|three|four|five|six|seven|eight|nine;
$teens = ten|eleven|twelve|thirteen|fourteen|fifteen|
         sixteen|seventeen|eighteen|nineteen;
$tens  = twenty|thirty|forty|fifty|sixty|seventy|
         eighty|ninety;

$channel     = $digit | $teens | $tens [$digit];
$device      = tv | dvd | vcr | stereo;
$power_state = on | off;
$cmd         = turn $power_state [the] $device |
               $device $power_state |
               [go] [to] channel $channel;

 ( < $cmd > )
```

Context-Free Grammars (CFGs)

- **Grammar is defined by,**
  - G = (V, T, P, S)
- **V : sets of non-terminals**
  - (e.g., NP,VP, …)
- **T : sets of terminals**
  - (e.g., mary, loves, …)
- **P : set of production rules**
  - (e.g., S → NP VP)
- **S : start symbol**



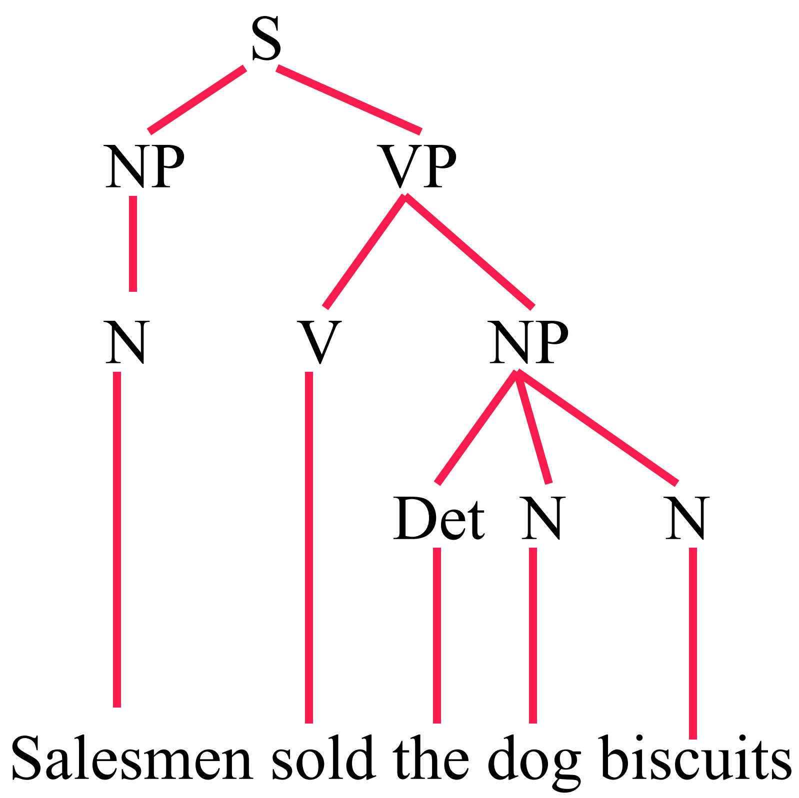
Context-Free Grammars (CFGs)

- **CFGs are more powerful than regular expression grammars**
- **Parsing Algorithm**
  - Searches through ways of combining grammatical rules
  - Generates a tree to illustrate the structure of the input sentence
  - “Parse Tree” records CFG rules
  - Top-down / Bottom-up Chart-Parsing approaches

Probabilistic Context-Free Grammars
(example from Josh Goodman's LM Tutorial)

Probabilities can be assigned to production rules (PCFGs)

S	→	NP VP	1.0
VP	→	V NP	0.5
VP	→	V NP NP	0.5
NP	→	Det N	0.5
NP	→	Det N N	0.5
N	→	salespeople	0.3
N	→	dog	0.4
N	→	biscuits	0.3
V	→	sold	1.0



Statistical Language Models

- **Want to estimate,**

$$P(\mathrm{W}) = P(\mathrm{w}_1 \mathrm{w}_2 \cdots \mathrm{w}_N)$$

- **Can decompose probability left-to-right**

$$\begin{aligned} P(\mathrm{W}) &= P(\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_N) \\ &= P(\mathrm{w}_1) P(\mathrm{w}_2 \mid \mathrm{w}_1) \cdots P(\mathrm{w}_N \mid \mathrm{w}_1, \mathrm{w}_2 \cdots \mathrm{w}_{N-1}) \\ &= \prod_{n=1}^{N} P(\mathrm{w}_n \mid \mathrm{w}_1, \mathrm{w}_2 \cdots \mathrm{w}_{n-1}) \end{aligned}$$

T-61.184

Statistical Language Model Example

$$P(\text{W}) = P(\text{center for spoken language research})$$

$$= P(\text{center})P(\text{for} \mid \text{center})\text{P}(\text{spoken} \mid \text{center for})\cdots$$

$$\cdots P(\text{research} \mid \text{center for spoken language})$$

- **Impossible to model the entire word sequence… never enough training data!**
- **Need to consider restricting the word-history used in computation of the probability estimate.**

“Markov Model” of Language

- **Cluster histories ending in same last N-1 words. “Markov Model” of Language.**
- **N=1** $$P(w_n \mid w_1, w_2 \cdots w_{n-1}) = P(w_n)$$
- **N=2** $$P(w_n \mid w_1, w_2 \cdots w_{n-1}) = P(w_n \mid w_{n-1})$$
- **N=3** $$P(w_n \mid w_1, w_2 \cdots w_{n-1}) = P(w_n \mid w_{n-1}, w_{n-2})$$

N-gram Language Model

- **N-gram models compute the probability of a word based on previous N-1 words:**
  - N=1 (Unigram)
  - N=2 (Bigram)
  - N=3 (Trigram)
- **Probabilities are estimated from a corpus of training data (text data).**
- **Once model is known, new sentences can be randomly generated by the model!**
- **Syntax roughly encoded by model, but ungrammatical and semantically “strange” sentences can be produced**

3-gram Example

... the united states of ???

$$P(\text{states} \mid \text{the united}) = \ldots$$

$$P(\text{of} \mid \text{united states}) = \ldots$$

$$P(\text{America} \mid \text{states of}) = \ldots$$

$$P(\text{Belgium} \mid \text{states of}) = \ldots$$

Estimating N-gram Probabilities

- **Given a text corpus, define the number of occurrences [count] of word (n) by,**

$$C(w_n)$$

- **Count of occurrences of word (n-1) followed by word (n),**

$$C(w_{n-1}, w_n)$$

- **And for 3 words,**

$$C(w_{n-2}, w_{n-1}, w_n)$$

Obtaining N-gram Probabilities

- **Maximum likelihood estimates of word probabilities are based on counting frequency of occurrence of word sequences from a training set of text data:**

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}$$

$$P(w_n \mid w_{n-2}, w_{n-1}) = \frac{C(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})}$$

2-gram Example

- **Assume the following Training Data:**

  `<s> John read her book </s>`

  `<s> I read a different book </s>`

  `<s> John read a book by Mulan </s>`

- **Calculate 2-gram:** `P(John read a book)`

$$P(\text{John} \mid <\text{s}>) = \frac{C(<\text{s}>, \text{John})}{C(<\text{s}>)} = \frac{2}{3} = 0.66$$

2-gram Example

$$P(\text{John} \mid <\text{s}>) = \frac{C(<\text{s}>, \text{John})}{C(<\text{s}>)} = \frac{2}{3}$$

$$P(\text{read} \mid \text{John}) = \frac{C(\text{John}, \text{read})}{C(\text{John})} = \frac{2}{2}$$

$$P(\text{a} \mid \text{read}) = \frac{C(\text{read}, \text{a})}{C(\text{read})} = \frac{2}{3} \qquad P(\text{book} \mid \text{a}) = \frac{C(\text{a}, \text{book})}{C(\text{a})} = \frac{1}{2}$$

$$P(</\text{s}> \mid \text{book}) = \frac{C(\text{book}, </\text{s}>)}{C(\text{book})} = \frac{2}{3}$$

$$P(\text{John read a book}) = P(\text{John} \mid <\text{s}>)P(\text{read} \mid \text{John}) \cdots P(</\text{s}> \mid \text{book})$$
$$\approx 0.14$$

“Unseen” Events in N-gram Models

- **Training Data:**

  `<s> John read her book </s>`

  `<s> I read a different book </s>`

  `<s> John read a book by Mulan </s>`

- **Calculate** `P(read | Mulan)` **:**

$$P(\text{read} \mid \text{Mulan}) = \frac{C(\text{Mulan}, \text{read})}{C(\text{Mulan})} = \frac{0}{1}$$

`P(Mulan read a book) = 0 !!!`

T-61.184

Making N-grams work for Speech Recognition

- **Raw probabilities estimates from N-grams can lead to 0 probability events (as we just saw)**

- **For a vocabulary of 20,000 words, there are 400 million possible bigrams. Given a corpus of 10 million training words, there will be MANY unseen events**

- **Methods for addressing this problem:**
  - Smoothing
  - Discounting
  - Backing-off
  - Interpolation

Smoothing

- **Adjust estimate of the probabilities to improve robustness to unseen data**
- **Allows non-zero probability to be assigned to all word strings**
- **Improves generalization of model**
- **Changes (Flattens) distribution:**
  - Low probability events become more likely
  - High probability events become less likely

“Add-1” Smoothing

- **Eliminates 0 probability problem by assuming n-gram occurs once more than it actually does: (V= vocabulary size)**

$$P(\mathrm{w}_n \mid \mathrm{w}_{n-2}, \mathrm{w}_{n-1}) = \frac{C(\mathrm{w}_{n-2}, \mathrm{w}_{n-1}, \mathrm{w}_n) + 1}{C(\mathrm{w}_{n-2}, \mathrm{w}_{n-1}) + V}$$

- **Does not work very well!**

Model Interpolation

- **Can interpolate statistics of lower-order counts to achieve higher-order n-gram probability,**

$$P(w_n \mid w_{n-2}, w_{n-1}) =$$
$$\lambda \frac{C(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})} + \mu \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} + (1 - \lambda - \mu) \frac{C(w_n)}{C(\bullet)}$$

- **Interpolation weights can be optimized on a held-out test set. Works so-so. Not used in practice…**

Good-Turing Smoothing

- **Redistribute the probability mass from "seen" events to "unseen" events, by discounting counts**
- **For any n-gram that appears "r" times, pretend that it appears "r*" times where:**

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

n_r : number of n-grams that occur exactly r times in the training data.

Good-Turing Smoothing Example

(From Josh Goodman's LM Tutorial)

- **Imagine you are fishing**
  - You catch 10 carp, 3 cod, 2 tuna, 1 trout, 1 salmon, 1 eel.
- **How likely is it that next species is new?**
  - 3/18 → use events seen once predict unseen event
- **How likely is it that next is tuna?**
  - Less than 2/18
  - Probabilities adjusted down to take into account unseen event

Good-Turing Smoothing Example

(From Josh Goodman's LM Tutorial)

- **10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel.**
- **How likely is new data (p_0).**
  Let n_1 be number occurring once (3), N be total (18). p_0=3/18

$$p_0 = \frac{n_1}{N}$$

- **How likely is eel? 1***

  n_1 =3, n_2 =1
  1^* =2 $\times$1/3 = 2/3
  P(eel) = 1^* /N = (2/3)/18 = 1/27

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

Katz's Discounting Model

- **Katz (1987): see reference on 2nd slide**
- **Count-dependent discounting factors applied to counts (r) which occur less than (k) times (k ≈ 7)**

$$r^* = r \cdot d_r$$

$$d_r = \frac{\left(\frac{(r+1)n_{r+1}}{rn_r}\right) - \left(\frac{(k+1)n_{k+1}}{n_1}\right)}{1 - \frac{(k+1)n_{k+1}}{n_1}} \qquad \text{if } (r < k)$$

- **d_r = 1 if r >= k.**

Katz (1987) Back-off Language Model

- **Uses Good-Turing Smoothing. "Back-off" to lower-order n-grams,**

$$P_{Katz}(w_n \mid w_{n-2}, w_{n-1}) =$$

$$\begin{cases} \dfrac{C^*(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})} & \text{if } C(w_{n-2}, w_{n-1}, w_n) > 0 \\ \alpha(w_{n-2}, w_{n-1}) \cdot P_{Katz}(w_n \mid w_{n-1}) & \text{otherwise} \end{cases}$$

- **α (back-off weight) is calculated so probabilities sum to 1**

Storage of Back-off N-gram Models

- **"ARPA" formatted language models are quite standardized in the speech community.**
  A back-off 3-gram model contains sets of,

- **Unigrams** $P(w_n)$ $\quad$ w_{n} $\quad$ $\alpha(w_n)$

- **Bigrams** $P(w_n | w_{n-1})$ $\quad$ $\mathrm{w}_{\mathrm{n\text{-}1}}\ \mathrm{w}_{\mathrm{n}}$ $\quad$ $\alpha(w_{n-1}, w_n)$

- **Trigrams** $P(w_n \mid w_{n\text{-}2}, w_{n-1})$ $\quad$ $\mathrm{w}_{\mathrm{n\text{-}2}}\ \mathrm{w}_{\mathrm{n\text{-}1}}\ \mathrm{w}_{\mathrm{n}}$

Computing a Probability from the Back-off (N=3)-gram Model

$$P(w_n \mid w_{n-2,} w_{n-1}) =$$

$$\begin{Bmatrix} P(w_n \mid w_{n-2,} w_{n-1}) & \text{if trigram exists, else,} \\ \alpha(w_{n\text{-}2}, w_{n-1}) P(w_n \mid w_{n-1}) & \text{if bigram } (w_{n\text{-}2}, w_{n-1}), \\ P(w_n \mid w_{n-1}) & \end{Bmatrix}$$

$$P(w_n \mid w_{n-1}) =$$

$$\begin{Bmatrix} P(w_n \mid w_{n-1}) & \text{if bigram exists,} \\ \alpha(w_{n\text{-}1}) P(w_n) & \text{otherwise} \end{Bmatrix}$$

Kneser-Ney Smoothing

- **Idea: weigh back-offs by number of contexts a word appears in.**
- **For example,**
  - “EggPlant Francisco” versus “Eggplant Stew”
  - P(Francisco | EggPlant) *versus* P(Stew | Eggplant)
  - “Francisco” is common, so back-off models favor this word
  - But “Francisco” tends to occur only in the context of “San” as in “San Francisco”.
  - “stew”, however, is common in many other contexts!
- **Details of Kneser-Ney are beyond this course, but this smoothing method tends to outperform all other methods**

Class n-gram Language Models

$$P(\mathrm{w}_n \mid \mathrm{w}_{n\text{-}1,} \mathrm{w}_{n-2}) = P(\mathrm{C}_n \mid \mathrm{C}_{n-1}, \mathrm{C}_{n-2}) P(\mathrm{w}_n \mid \mathrm{C}_n)$$

"I'd like to fly on <u>American</u>",
"I'd like to fly on <u>Delta</u>"

"I'd like to fly on [airline_company]"

[airline_company]	$P(\mathrm{w}_n \mid \mathrm{C}_n)$
AMERICAN	0.3000
DELTA	0.7000

T-61.184

Example Class Types for a Spoken Dialog System for Air Travel

■ Months	{January, February, March…}
■ Days of Week	{Monday, Tuesday, …}
■ Year	{2003, 2004…}
■ Time Period	{morning, afternoon,…}
■ Hour Number	{one, two, three, … twelve}
■ Minute Number	{fifteen, forty_five, thirty}
■ Ordinal Number	{first, second, third, fourth…}
■ Cardinal Number	{one, two, three, four…}
■ City	{Denver, Boston, Helsinki…}

How to Derive the Classes?

- **Design them by Hand**
  - Useful for spoken dialog systems
  - Requires that you tag your training text by class labels. Some labels are ambiguous!
    - [one] day, I'd like to go to Colorado
    - I'll take option [number:one]
- **Use syntactic class labels (e.g., part of speech tags)**
- **Automatic clustering approach**
  - Design word classes to minimize entropy in the language model
  - Swap words in and out of classes and test entropy.
  - Implemented in the Cambridge HTK toolkit

Evaluating Language Models

- **How can we tell a good language model from a bad language model?**

- **Can evaluate LMs by measuring Word Error Rate (WER), but this requires running the speech recognizer**

- **Alternately, we can look at the probability that the LM assigns to word sequences**

Language Model Perplexity

- Corpus perplexity is defined as,

$$PP(\mathrm{W}) = P(w_1 \cdots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\prod_{n=1}^{N} \frac{1}{P(w_n \mid w_{1\ldots n-1})}}$$

- Corpus perplexity is the geometric average of the reciprocal probability over all N words.
- Therefore, minimizing corpus perplexity is the same as maximizing the conditional likelihood

Another view of Perplexity

- **Expanding the definition and taking the logarithm,**

$$\log PP(\mathrm{W}) = -\frac{1}{N}\sum_{n=1}^{N}\log P(\mathrm{w}_n \mid \mathrm{w}_1 \cdots \mathrm{w}_{n-1})$$

- **Log-PP(W) is referred to as <u>entropy</u>. Average number of bits-per-word required to encode the test material using this language model and an optimal coder.**

Notes on Perplexity

- **Generally computed on a held-out test set**
- **Can be interpreted as the average number of word choices during the recognition process**
- **Smaller the perplexity, the lower the expected error rate of the recognizer (not always true!)**
- **Does not take acoustic confusability into account.**

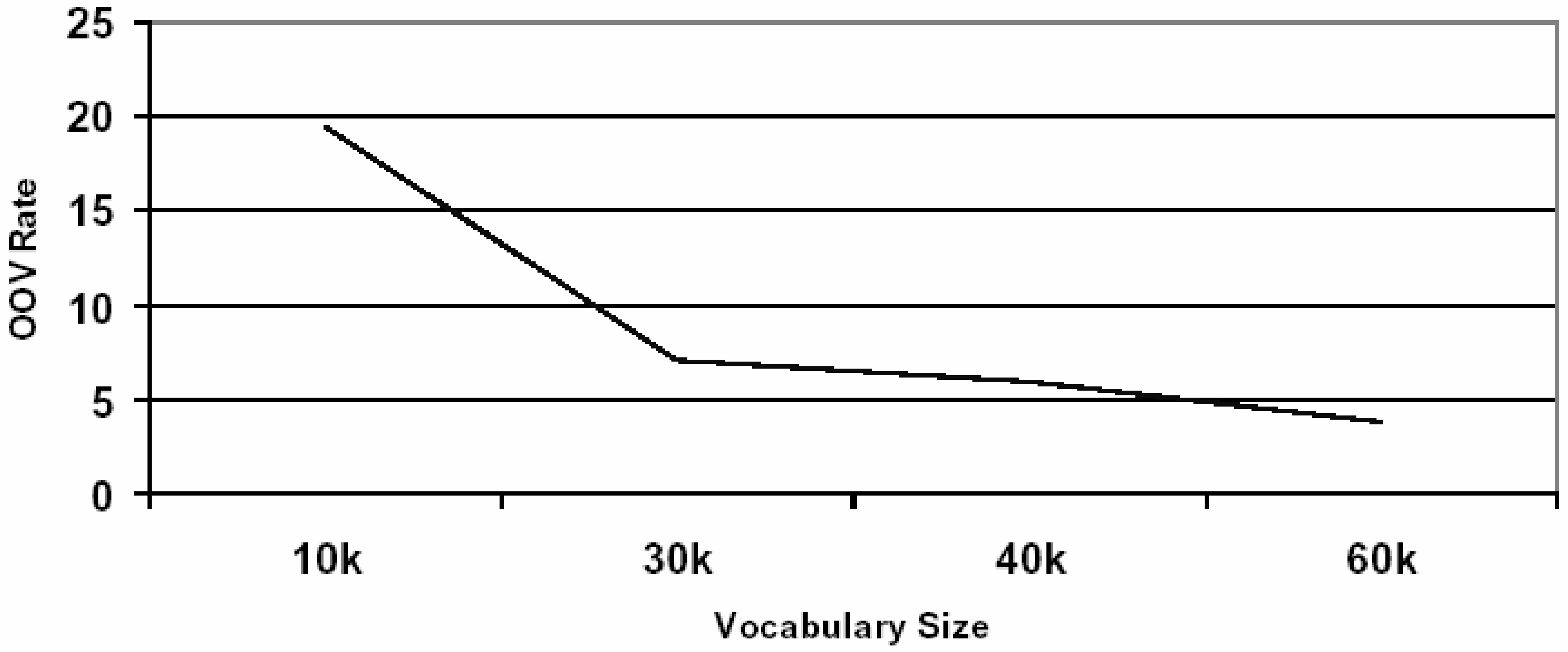
Typical Perplexity Values

- Lower PP(W) sometimes reflects less confusion in speech recognition
- Typical PP(W) for English: 50 → 1000
- 5k vocabulary read financial (WSJ) newspaper text: trigram PP(W) = 128, bigram PP(W) = 176.
- 2k vocabulary ATIS (Air Travel Information System) Task: PP(W) = less than 20.

Out of Vocabulary Words (OOV)

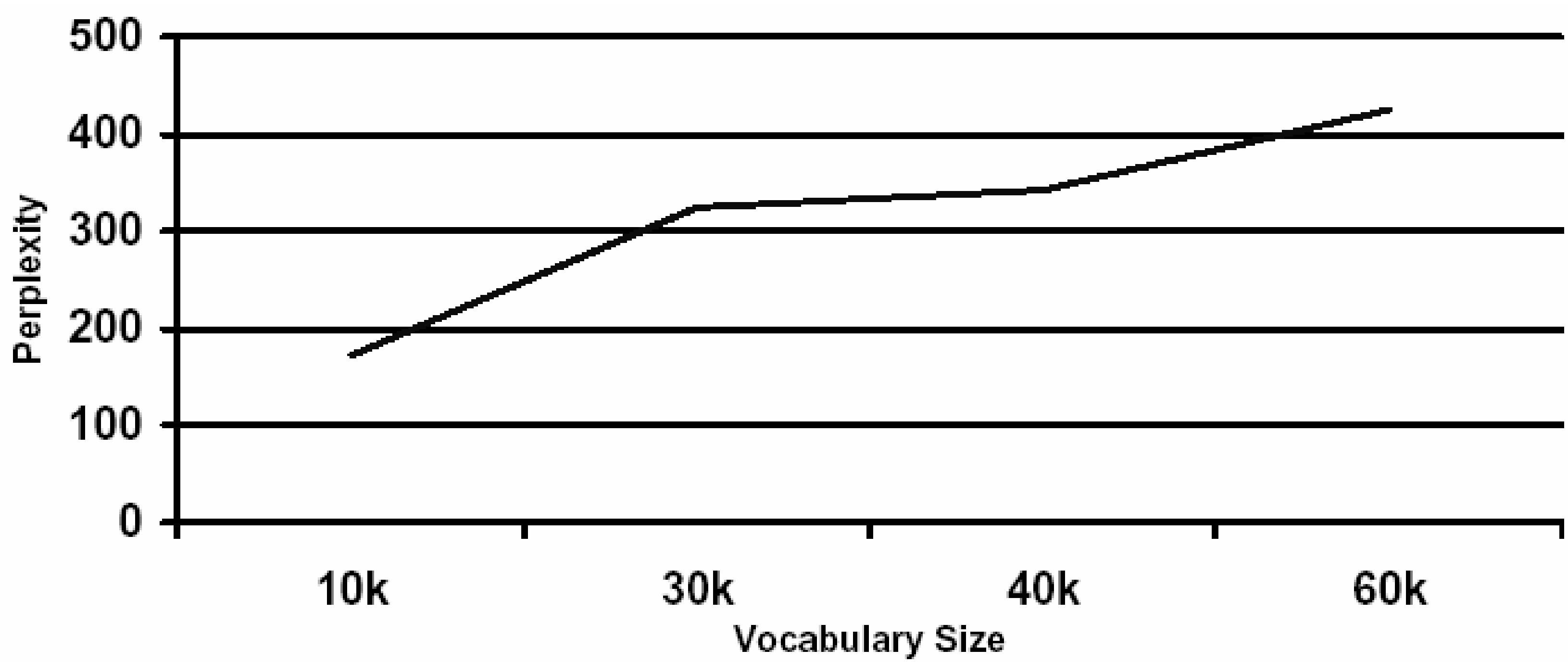
- OOV Words: are words that are not part of the recognizer vocabulary, but may be spoken by the user
- We need about a 200,000 word vocabulary to cover 99.5% of the words in spoken English
- Large Vocabulary ASR systems have 64k word vocabularies (20k-64k typical).

OOV Rate vs. Vocabulary Size



T-61.184

Perplexity vs. Vocabulary Size



- **Bigram language model; 500 million word corpus**

Performance of Smoothed N-gram LMs with Katz Backoff

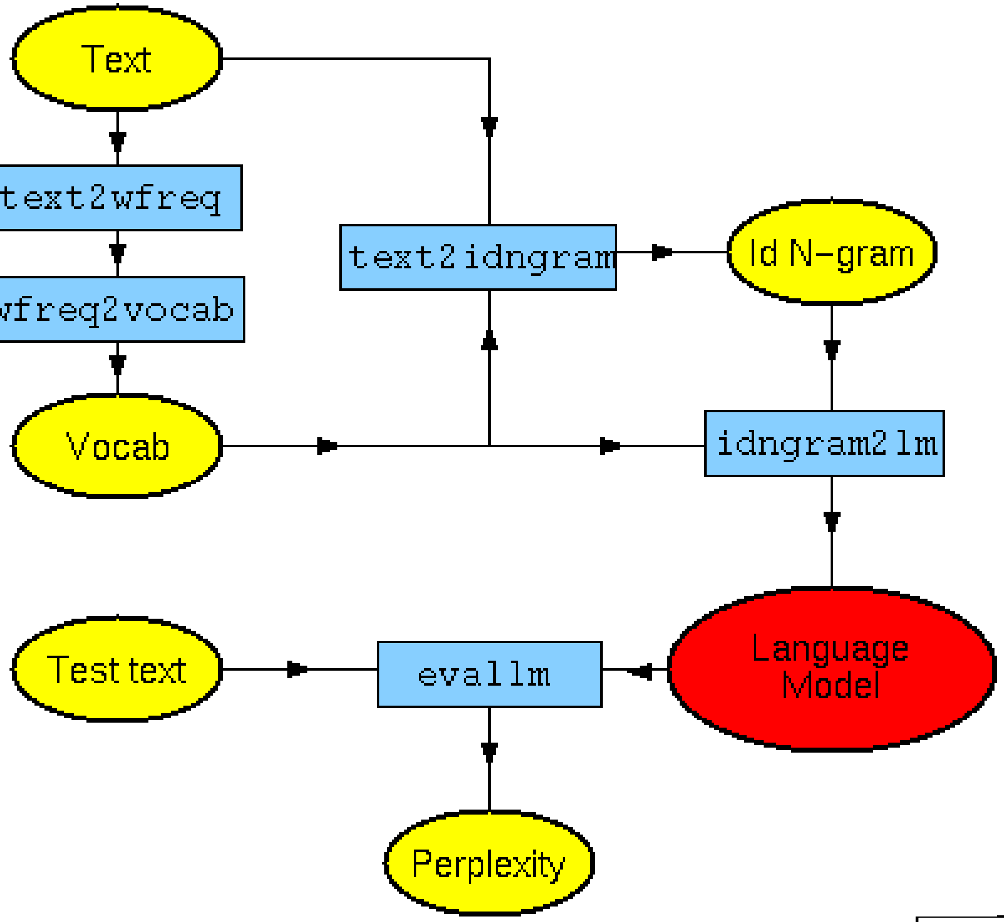
Back-off Model	Test-Set Perplexity	Word Error Rate
Unigram	1196	14.8%
Bigram	176	11.4%
Trigram	95	9.7%

- 60k vocabulary dictation application
- 260 million word text training corpus
- Microsoft Whisper Speech Recognizer

Software for Estimating and Evaluating N-gram Language Models

- **CMU/Cambridge Statistical LM toolkit**
  - http://www.speech.cs.cmu.edu/SLM_info.html
  - **Easy to use**
  - **Implements basic smoothing techniques**
  - **Restricted to 64k vocabulary or smaller**
- **SRI Statistical Language Model Toolkit**
  - http://www.speech.sri.com/projects/srilm/
  - **More advanced algorithms implemented**
  - **Can interpolate language models for example**

CMU/Cambridge Statistical LM toolkit



CMU/Cambridge Toolkit Example

- **Example training data placed in file: `training.txt`**

- **Use <s> and </s> to mark the begin and end of sentences (context cues)**

```
<s> I WANT TO GO FROM DENVER TO BOSTON TOMORROW MORNING </s>
<s> I WOULD LIKE TO GO TO SEATTLE THIS AFTERNOON </s>
<s> DEPARTING SEATTLE </s>
<s> IN THE LATE MORNING </s>
<s> LATE MORNING GOING TO DENVER FROM BOSTON </s>
<s> FROM DENVER LEAVING IN THE MORNING ARRIVING IN BOSTON </s>
```

Using the CMU/Cambridge Toolkit

- <u>**Step 1**</u>**: Determine the vocabulary for the task**

```
cat training.txt | text2wfreq  > vocab.freq
cat vocab.freq   | wfreq2vocab > training.vocab
```

- `training.vocab` **now contains the vocabulary for your data. You can edit this file to remove words from the system vocabulary.**

Using the CMU/Cambridge Toolkit

- **Step 2: put all context cues into a file,**

```
echo "<s>" > training.ccs
```

- **Step 3: convert N-grams to a list of word ID's**

```
cat training.txt | text2idngram -n 3 \
       -vocab training.vocab \
       -buffer 100 \
       -temp /usr/tmp > training.id3gram
```

-n 3 : implies back-off 3-gram language model

Using the CMU/Cambridge Toolkit

- **<u>Step 4</u>: convert N-grams in word ID format to a back-off language model**

```
idngram2lm -idngram training.id3gram \
           -vocab training.vocab \
           -arpa training.arpa \
           -cutoffs 0 0 \
           -good_turing \
           -context training.ccs
```

- **Output model will be in** `training.arpa`

Example Back-off N-gram Language Model Format

```
Beginning of data mark: \data\

ngram 1=nr              # number of 1-grams
ngram 2=nr              # number of 2-grams
ngram 3=nr              # number of 3-grams

\1-grams:
p_1     wd_1 bo_wt_1
\2-grams:
p_2     wd_1 wd_2 bo_wt_2
\3-grams:
p_3     wd_1 wd_2 wd_3
end of data mark: \end\
```

T-61.184

Understanding the Output file format

```
ngram 1=23
ngram 2=41
ngram 3=50

\1-grams:
-1.0086 </s>     -0.7238
-0.9294 <s>      -0.2369
-1.7076 AFTERNOON        -0.2562
-1.7076 ARRIVING         -0.2747
-1.4065 BOSTON   -0.2467
-1.2304 DENVER   -0.2167
-1.7076 DEPARTING        -0.2837
```

23 words in vocabulary; 41 bigrams, 50 trigrams

log(p_1)

log (bo_wt_1)

Computing P(W) using a Back-off Language Model

```
p(wd3|wd1,wd2)= if(trigram exists)
                     p_3(wd1,wd2,wd3)
                else if(bigram w1,w2 exists)
                     bo_wt_2(w1,w2)*p(wd3|wd2)
                else p(wd3|w2)

p(wd2|wd1)    = if(bigram exists)
                      p_2(wd1,wd2)
                else
                      bo_wt_1(wd1)*p_1(wd2)
```

REMEMBER!! Values in ARPA file are in log scale. Replace multiplies with additions

Evaluating Test-Set Perplexity

- **Use** `evallm` **program included in CMU/Cambridge Toolkit**
- `evallm -arpa training.arpa`
  `evallm: perplexity -text mytest.txt`
- **Where** `mytest.txt` **contains test sentences**

Next Week

- **Some talk about the lexicon and predicting word pronunciations from letter sequences…**
- **Search methods for large vocabulary speech recognition**