Objectives

Introduction:
Noisy Channel Model
Chomsky Hierarchy
Motivation

Techniques:
Simple
Generalized Interpolation
Deleted Interpolation
Good-Turing Estimates
Katz Smoothing
Kneser-Ney Bigram Smoothing
Class N-grams

On-Line Resources:
LM Overview
Ngram Smoothing
Turing Intro
Good Turing Smoothing

● Objectives:
  ○ Why do we need N-gram smoothing?
  ○ Deleted interpolation
  ○ Backoff language models
  ○ Discounting

This lecture combines material from the course textbook:


and from this source:

A noisy communication theory model for speech production and perception:

- **Message Source** → **Linguistic Channel** → **Articulatory Channel** → **Acoustic Channel**

  Observable: Message → Words → Phones → Features

Bayesian formulation for speech recognition:

\[ P(W|A) = P(A|W)P(W)/P(A) \]

Objective: minimize the word error rate by maximizing \( P(W|A) \)

Approach: maximize \( P(A|W) \) (training)

Components:
- \( P(A|W) \): acoustic model (hidden Markov models, mixture of Gaussians)
- \( P(W) \): language model (statistical, N-grams, finite state networks)
- \( P(A) \): acoustics (ignore during maximization)

The language model typically predicts a small set of next words based on knowledge of a finite number of previous words (N-grams) — leads to search space reduction.
We can categorize language models by their generative capacity:

<table>
<thead>
<tr>
<th>Type of Grammar</th>
<th>Constraints</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phrase Structure</td>
<td>A -&gt; B</td>
<td>Turing Machine (Unrestricted)</td>
</tr>
<tr>
<td>Context Sensitive</td>
<td>aAb -&gt; aBb</td>
<td>Linear Bounded Automata (N-grams, Unification)</td>
</tr>
<tr>
<td>Context Free</td>
<td>A -&gt; B Constraint: A is a non-terminal.</td>
<td>Push down automata (JSGF, RTN, Chart Parsing)</td>
</tr>
<tr>
<td></td>
<td>Equivalent to:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A -&gt; w</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A -&gt; BC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>where &quot;w&quot; is a terminal;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B,C are non-terminals (Chomsky normal form)</td>
<td></td>
</tr>
<tr>
<td>Regular</td>
<td>A -&gt; w</td>
<td>Finite-state automata (Network decoding)</td>
</tr>
<tr>
<td></td>
<td>A -&gt; wB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Subset of CFG)</td>
<td></td>
</tr>
</tbody>
</table>

- CFGs offer a good compromise between parsing efficiency and representational power.
- CFGs provide a natural bridge between speech recognition and natural language processing.
WHY IS SMOOTHING SO IMPORTANT?

- A key problem in N-gram modeling is the inherent data sparseness.

- For example, in several million words of English text, more than 50% of the trigrams occur only once; 80% of the trigrams occur less than five times (see SWB data also).

- Higher order N-gram models tend to be domain or application specific. Smoothing provides a way of generating generalized language models.

- If an N-gram is never observed in the training data, can it occur in the evaluation data set?

- Solution: **Smoothing** is the process of flattening a probability distribution implied by a language model so that all reasonable word sequences can occur with some probability. This often involves broadening the distribution by redistributing weight from high probability regions to zero probability regions.
SMOOTHING IS AN INTUITIVELY SIMPLE CONCEPT

- **Simple smoothing**: pretend each bigram occurs once more than it actually does in the training data set

\[
P(w_i | w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{\sum_{w_i} (1 + C(w_{i-1}, w_i))} = \frac{1 + C(w_{i-1}, w_i)}{V + \sum_{w_i} C(w_{i-1}, w_i)}
\]

- Note that the probability density function must be balanced to that it still sums to one.
THE BACKOFF MODEL: A FLEXIBLE TRADE-OFF BETWEEN ACCURACY AND COMPLEXITY

- **Backoff smoothing**: Approximate the probability of an unobserved N-gram using more frequently occurring lower order N-grams

\[
P_{\text{smooth}}(w_i|w_{i-n+1} \cdots w_{i-1}) =
\begin{cases} 
\alpha(w_i|w_{i-n+1} \cdots w_{i-1}) & C(w_{i-n+1} \cdots w_i) > 0 \\
\gamma(w_{i-n+1} \cdots w_{i-1})P_{\text{smooth}}(w_i|w_{i-n+2} \cdots w_{i-1}) & C(w_{i-n+1} \cdots w_i) = 0
\end{cases}
\]

- If an N-gram count is zero, we approximate its probability using a lower order N-gram.
- The scaling factor is chosen to make the conditional distribution sum to one.
- Extremely popular for N-gram modeling in speech recognition because you can control complexity as well as generalization.
DELETED INTERPOLATION SMOOTHING

- We can linearly interpolate a bigram and a unigram model as follows:

\[ P_I(w_i | w_{i-1}) = \lambda P(w_i | w_{i-1}) + (1 - \lambda) P(w_i) \]

- We can generalize this to interpolating an N-gram model using and (N-1)-gram model:

\[
P_I(w_i | w_{i-n+1} \ldots w_{i-1}) = \lambda P(w_i | w_{i-n+1} \ldots w_{i-1}) + (1 - \lambda) P(w_i | w_{i-n+2} \ldots w_{i-1})
\]

Note that this leads to a recursive procedure if the lower order N-gram probability also doesn't exist. If necessary, everything can be estimated in terms of a unigram model.

- A scaling factor is used to make sure that the conditional distribution will sum to one.

- An N-gram specific weight is used. In practice, this would lead to far too many parameters to estimate. Hence, we need to cluster such weights (by word class perhaps), or in the extreme, use a single weight.

- The optimal value of the interpolation weight can be found using Baum's reestimation algorithm. However, Bahl et al suggest a simpler procedure that produces a comparable result. We demonstrate the procedure here for the case of a bigram language model:

1. Divide the total training data into kept and held-out data sets.
2. Compute the relative frequency for the bigram and the unigram from the kept data.
3. Compute the count for the bigram in the held-out data set.
4. Find a weight by maximizing the likelihood:

\[
\sum_{N(v) \in R} \sum w_2 C(v, w_2) \log(\lambda \times f(w_2 | w_1) + (1 - \lambda) f(w_2 | w_1))
\]

This is equivalent to solving this equation:

\[
\sum_{N(v) \in R} \sum w_2 N(v, w_2) \left[ \lambda + \frac{f(w_2 | v)}{f(w_3) - f(w_2 | v)} \right]^{-1} = 0
\]
The Good-Turing estimate states that for any N-gram, $\alpha_r$, that occurs $r$ times, we should reestimate this frequency of occurrence as:

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

where $n_r$ is the number of N-grams that occur exactly $r$ times. This count can be converted to a probability by dividing by the total number of N-gram tokens:

$$P(\alpha) = \frac{r^*}{N}$$

where

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

The justification for this equation is as follows:

Suppose we have training data for N-grams $\alpha_1 \ldots \alpha_s$. Let $c(\alpha_j)$ denote the number of times the N-gram $\alpha_j$ occurs in the training data, and $p_j$ be the true probability of $\alpha_j$.

Estimating $p_j$ by using its frequency of occurrence can be expanded as:

$$E(p_j|c(\alpha_j) = r) = \sum_{k=1}^{s} p(i = k|c(\alpha_j) = r)p_k$$

We are using the chance that a randomly selected N-gram, $\alpha_j$, with count $r$, is in fact $\alpha_k$. This can be rewritten into:

$$p(i = k|c(\alpha_j) = r) = \frac{p(c(\alpha_j) = r)}{\sum_{i=1}^{s} p(c(\alpha_j) = r)} = \frac{\binom{N}{r} p_j (1-p_j)^{N-r}}{\sum_{l=1}^{s} \binom{N}{l} p_l (1-p_l)^{N-r}}$$

Substituting this into our expression for $E(p_j|c(\alpha_j) = r)$:

$$E(p_j|c(\alpha_j) = r) = \frac{\sum_{l=1}^{s} \binom{N}{l} p_l (1-p_l)^{N-r}}{\sum_{l=1}^{s} \binom{N}{l} p_l (1-p_l)^{N-r}} p_k$$

Noting that every N-gram token counts as 1, we can express the expected value of $n_r$ as:

$$E(n_r) = \sum_{i=1}^{s} p(c(\alpha_j) = r) = \sum_{l=1}^{s} \binom{N}{l} p_l (1-p_l)^{N-r}$$

We can show that:

$$\frac{r+1}{N+1} E_n(\alpha_{r+1}) = E(p_j|c(\alpha_j) = r)$$

We can make the approximation that $n_r = E_n(\alpha_r)$ and

$$n_{r+1} = \frac{N}{N+1} E_n(\alpha_{r+1})$$

Combining these results:

$$P(\alpha) = \frac{r^*}{N}$$

$$r^* = NP(\alpha)$$

$$= NE(p_j|c(\alpha_j) = r)$$

$$= N \frac{r+1}{N+1} E_n(\alpha_{r+1})$$

$$= (r + 1) \frac{n_{r+1}}{N}$$

Note that we must pre-smooth the distribution so $n_r > 0$. 

$$n_r = \frac{N}{N+1} E_n(\alpha_r)$$
KATZ SMOOTHING BASED ON GOOD-TURING ESTIMATES

- **Katz smoothing** applies Good-Turing estimates to the problem of backoff language models.

- Katz smoothing uses a form of *discounting* in which the amount of discounting is proportional to that predicted by the Good-Turing estimate.

- The total number of counts discounted in the global distribution is equal to the total number of counts that should be assigned to N-grams with zero counts according to the Good-Turing estimate (preserving the unit area constraint for the pdf).

- Katz Smoothing:

\[
P_{Katz}(w_i | w_{i-1}) = \begin{cases} 
  \frac{C(w_{i-1}w_i)}{C(w_{i-1})}, & r > k \\
  d_r \frac{C(w_{i-1}w_i)}{C(w_{i-1})}, & k \geq r > 0 \\
  \alpha(w_{i-1}) P(w_i), & r = 0 
\end{cases}
\]

where 
\[
d_r = \frac{r^* - (k+1)n_k + 1}{r - \frac{(k+1)n_k + 1}{n_1}} \quad \text{and} \quad \alpha(w_{i-1}) = \frac{1 - \sum_{w_i : r > 0} P_{Katz}(w_i | w_{i-1})}{1 - \sum_{w_i : r > 0} P_{Katz}(w_i)}.
\]
- **Absolute discounting** involves subtracting a fixed discount, \( D \), from each nonzero count, and redistributing this probability mass to N-grams with zero counts.

- We implement absolute discounting using an interpolated model:

\[
P_{\text{abs}}(w_i | w_{i-n+1} \ldots w_{i-1}) = \frac{\max\{C(w_{i-n+1} \ldots w_i) - D, 0\}}{\sum_{w_i} C(w_{i-n+1} \ldots w_i)} + (1 - \lambda_{w_{i-n+1} \ldots w_{i-1}}) P_{\text{abs}}(w_i | w_{i-n+2} \ldots w_{i-1})
\]

- **Kneser-Ney smoothing** combines notions of discounting with a backoff model. Here is an algorithm for bigram smoothing:

\[
P_{KN}(w_i | w_{i-1}) = \begin{cases} \max\{C(w_{i-1} w_i) - D, 0\} / C(w_{i-1}) & C(w_{i-1} w_i) > 0 \\ \alpha(w_{i-1}) P_{KN}(w_i) & \text{otherwise} \end{cases}
\]

where

\[
P_{KN}(w_i) = \frac{C(w_i \cdot)}{\sum_{w_i} C(w_i \cdot)}
\]

and \( C(w_i \cdot) \) is the number of unique words preceding \( w_i \).

\( \alpha(w_{i-1}) \) is chosen to make the distribution sum to 1:

\[
\alpha(w_{i-1}) = \frac{1 - \sum_{w_i: C(w_i \cdot) > 0} \frac{\max\{C(w_{i-1} w_i) - D, 0\}}{C(w_{i-1})}}{1 - \sum_{w_i: C(w_i \cdot) > 0} P_{KN}(w_i)}
\]

- Kneser-Ney smoothing constructs a lower order distribution that is consistent with the smoothed higher order distribution.
CLASS N-GRAMS

- Recall we previously discussed defining equivalence classes for words that exhibit similar semantic and grammatical behavior.

- Class based language models have been shown to be effective for reducing memory requirements for real-time speech applications, and supporting rapid adaption of language models.

- A word probability can be conditioned on the previous N-1 word classes:

\[ P(w_i | c_{i-n+1} \cdots c_i) = P(w_i | c_i) P(c_i | c_{i-n+1} \cdots c_i) \]

- We can express the probability of a word sequence in terms of class N-grams:

\[ P(W) = \sum_{c_1 \cdots c_n} \prod_i P(w_i | c_i) P(c_i | c_{i-n+1} \cdots c_i) \]

- If the classes are non-overlapping:

\[ P(W) = \prod_i P(w_i | c_i) P(c_i | c_{i-n+1} \cdots c_i) \]

- If we consider the case of a bigram language model, we can derive a simple estimate for a bigram probability in terms of word and class counts:

\[ P(w_i | w_{i-1}) \approx P(w_i | c_{i-1}) = P(w_i | c_i) P(c_i | c_{i-1}) \]

\[ = \frac{C(w_i c_i) C(c_{i-1} c_i)}{C(c_i) C(c_{i-1})} \]

\[ = \frac{C(w_i) C(c_{i-1} c_i)}{C(c_i) C(c_{i-1})} \]

- Class N-grams have not provided significant improvements in performance, but have provided a simple means of integrating linguistic knowledge and data-driven statistical knowledge.