Chapter 7

Language models

Statistical Machine Translation
Language models

- **Language models** answer the question:
  
  *How likely is a string of English words good English?*

- Help with reordering

  \[ p_{LM}(\text{the house is small}) > p_{LM}(\text{small the is house}) \]

- Help with word choice

  \[ p_{LM}(\text{I am going home}) > p_{LM}(\text{I am going house}) \]
N-Gram Language Models

- Given: a string of English words \( W = w_1, w_2, w_3, ..., w_n \)
- Question: what is \( p(W) \)?
- Sparse data: Many good English sentences will not have been seen before

\[ p(w_1, w_2, w_3, ..., w_n) = p(w_1) p(w_2|w_1) p(w_3|w_1, w_2)...p(w_n|w_1, w_2, ...w_{n-1}) \]

(not much gained yet, \( p(w_n|w_1, w_2, ...w_{n-1}) \) is equally sparse)
Markov Chain

- **Markov assumption:**
  - only previous history matters
  - limited memory: only last $k$ words are included in history
    (older words less relevant)

$\rightarrow$ $k$:th order Markov model

- For instance 2-gram language model:

  $$p(w_1, w_2, w_3, \ldots, w_n) \approx p(w_1) \ p(w_2|w_1) \ p(w_3|w_2) \ldots p(w_n|w_{n-1})$$

- What is conditioned on, here $w_{i-1}$ is called the **history**
Estimating N-Gram Probabilities

- Maximum likelihood estimation

\[ p(w_2 | w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \]

- Collect counts over a large text corpus

- Millions to billions of words are easy to get

  (trillions of English words available on the web)
Example: 3-Gram

- Counts for trigrams and estimated word probabilities

<table>
<thead>
<tr>
<th>The green (total: 1748)</th>
<th>The red (total: 225)</th>
<th>The blue (total: 54)</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>c.</td>
<td>prob.</td>
</tr>
<tr>
<td>paper</td>
<td>801</td>
<td>0.458</td>
</tr>
<tr>
<td>group</td>
<td>640</td>
<td>0.367</td>
</tr>
<tr>
<td>light</td>
<td>110</td>
<td>0.063</td>
</tr>
<tr>
<td>party</td>
<td>27</td>
<td>0.015</td>
</tr>
<tr>
<td>ecu</td>
<td>21</td>
<td>0.012</td>
</tr>
</tbody>
</table>

- 225 trigrams in the Europarl corpus start with the red
- 123 of them end with cross
→ maximum likelihood probability is \( \frac{123}{225} = 0.547 \).
How good is the LM?

• A good model assigns a text of real English \( W \) a high probability

• This can be also measured with cross entropy:

\[
H(W) = \frac{1}{n} \log p(W_1^n)
\]

• Or, perplexity

\[
\text{perplexity}(W) = 2^{H(W)}
\]
Example: 4-Gram

<table>
<thead>
<tr>
<th>prediction</th>
<th>$p_{LM}$</th>
<th>$-\log_2 p_{LM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{LM}(i</td>
<td>&lt;s&gt;&lt;s&gt;)$</td>
<td>0.109</td>
</tr>
<tr>
<td>$p_{LM}(\text{would}</td>
<td>&lt;s&gt;i)$</td>
<td>0.144</td>
</tr>
<tr>
<td>$p_{LM}(\text{like}</td>
<td>i \text{ would})$</td>
<td>0.489</td>
</tr>
<tr>
<td>$p_{LM}(\text{to}</td>
<td>\text{would like})$</td>
<td>0.905</td>
</tr>
<tr>
<td>$p_{LM}(\text{commend}</td>
<td>\text{like to})$</td>
<td>0.002</td>
</tr>
<tr>
<td>$p_{LM}(\text{the}</td>
<td>\text{to commend})$</td>
<td>0.472</td>
</tr>
<tr>
<td>$p_{LM}(\text{rapporteur}</td>
<td>\text{commend the})$</td>
<td>0.147</td>
</tr>
<tr>
<td>$p_{LM}(\text{on}</td>
<td>\text{the rapporteur})$</td>
<td>0.056</td>
</tr>
<tr>
<td>$p_{LM}(\text{his}</td>
<td>\text{rapporteur on})$</td>
<td>0.194</td>
</tr>
<tr>
<td>$p_{LM}(\text{work}</td>
<td>\text{on his})$</td>
<td>0.089</td>
</tr>
<tr>
<td>$p_{LM}(.\text{</td>
<td>his work})$</td>
<td>0.290</td>
</tr>
<tr>
<td>$p_{LM}(&lt;/s&gt;</td>
<td>\text{work .})$</td>
<td>0.99999</td>
</tr>
<tr>
<td>average</td>
<td>2.634</td>
<td></td>
</tr>
</tbody>
</table>
## Comparison 1–4-Gram

<table>
<thead>
<tr>
<th>word</th>
<th>unigram</th>
<th>bigram</th>
<th>trigram</th>
<th>4-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6.684</td>
<td>3.197</td>
<td>3.197</td>
<td>3.197</td>
</tr>
<tr>
<td>would</td>
<td>8.342</td>
<td>2.884</td>
<td>2.791</td>
<td>2.791</td>
</tr>
<tr>
<td>like</td>
<td>9.129</td>
<td>2.026</td>
<td>1.031</td>
<td>1.290</td>
</tr>
<tr>
<td>to</td>
<td>5.081</td>
<td>0.402</td>
<td>0.144</td>
<td>0.113</td>
</tr>
<tr>
<td>commend</td>
<td>15.487</td>
<td>12.335</td>
<td>8.794</td>
<td>8.633</td>
</tr>
<tr>
<td>the</td>
<td>3.885</td>
<td>1.402</td>
<td>1.084</td>
<td>0.880</td>
</tr>
<tr>
<td>rapporteur</td>
<td>10.840</td>
<td>7.319</td>
<td>2.763</td>
<td>2.350</td>
</tr>
<tr>
<td>on</td>
<td>6.765</td>
<td>4.140</td>
<td>4.150</td>
<td>1.862</td>
</tr>
<tr>
<td>his</td>
<td>10.678</td>
<td>7.316</td>
<td>2.367</td>
<td>1.978</td>
</tr>
<tr>
<td>work</td>
<td>9.993</td>
<td>4.816</td>
<td>3.498</td>
<td>2.394</td>
</tr>
<tr>
<td>.</td>
<td>4.896</td>
<td>3.020</td>
<td>1.785</td>
<td>1.510</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>4.828</td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>average</td>
<td>8.051</td>
<td>4.072</td>
<td>2.634</td>
<td>2.251</td>
</tr>
<tr>
<td>perplexity</td>
<td>265.136</td>
<td>16.817</td>
<td>6.206</td>
<td>4.758</td>
</tr>
</tbody>
</table>

Chapter 7: Language Models
Unseen N-Grams

- We have seen *i like to* in our corpus
- We have never seen *i like to smooth* in our corpus

→ $p(\text{smooth}|i \text{ like to}) = 0$

- Any sentence that includes *i like to smooth* will be assigned probability 0
Add-One Smoothing

• For all possible n-grams, add the count of one.

\[ p = \frac{c + 1}{n + v} \]

- \( c \) = count of n-gram in corpus
- \( n \) = count of history
- \( v \) = vocabulary size

• But there are many more unseen n-grams than seen n-grams

• Example: Europarl 2-bigrams:
  - 86,700 distinct words
  - \( 86,700^2 = 7,516,890,000 \) possible bigrams
  - but only about 30,000,000 words (and bigrams) in corpus
Add-$\alpha$ Smoothing

- Add $\alpha < 1$ to each count

\[ p = \frac{c + \alpha}{n + \alpha v} \]

- What is a good value for $\alpha$?

- Could be optimized on held-out set
### Example: 2-Grams in Europarl

<table>
<thead>
<tr>
<th>Count</th>
<th>Adjusted count</th>
<th>Test count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>((c + 1)\frac{n}{n + v^2})</td>
<td>((c + \alpha)\frac{n}{n + \alpha v^2})</td>
</tr>
<tr>
<td>0</td>
<td>0.00378</td>
<td>0.00016</td>
</tr>
<tr>
<td>1</td>
<td>0.00755</td>
<td>0.95725</td>
</tr>
<tr>
<td>2</td>
<td>0.01133</td>
<td>1.91433</td>
</tr>
<tr>
<td>3</td>
<td>0.01511</td>
<td>2.87141</td>
</tr>
<tr>
<td>4</td>
<td>0.01888</td>
<td>3.82850</td>
</tr>
<tr>
<td>5</td>
<td>0.02266</td>
<td>4.78558</td>
</tr>
<tr>
<td>6</td>
<td>0.02644</td>
<td>5.74266</td>
</tr>
<tr>
<td>8</td>
<td>0.03399</td>
<td>7.65683</td>
</tr>
<tr>
<td>10</td>
<td>0.04155</td>
<td>9.57100</td>
</tr>
<tr>
<td>20</td>
<td>0.07931</td>
<td>19.14183</td>
</tr>
</tbody>
</table>

- Add-\(\alpha\) smoothing with \(\alpha = 0.00017\)
- \(t_c\) are average counts of n-grams in test set that occurred \(c\) times in corpus
Deleted Estimation

• Estimate true counts in held-out data
  – split corpus in two halves: training and held-out
  – counts in training $C_t(w_1, ..., w_n)$
  – number of ngrams with training count $r$: $N_r$
  – total times ngrams of training count $r$ seen in held-out data: $T_r$

• Held-out estimator:

$$p_h(w_1, ..., w_n) = \frac{T_r}{N_r N} \quad \text{where } \text{count}(w_1, ..., w_n) = r$$

• Both halves can be switched and results combined

$$p_h(w_1, ..., w_n) = \frac{T_r^1 + T_r^2}{N(N_r^1 + N_r^2)} \quad \text{where } \text{count}(w_1, ..., w_n) = r$$
Good-Turing Smoothing

• Adjust actual counts $r$ to expected counts $r^*$ with formula

$$r^* = (r + 1) \frac{N_{r+1}}{N_r}$$

- $N_r$ number of n-grams that occur exactly $r$ times in corpus
- $N_0$ total number of n-grams
### Good-Turing for 2-Grams in Europarl

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

The adjusted count is fairly accurate when compared against the test count.
Derivation of Good-Turing

- A specific n-gram $\alpha$ occurs with (unknown) probability $p$ in the corpus
- Assumption: all occurrences of an n-gram $\alpha$ are independent of each other
- Number of times $\alpha$ occurs in corpus follows binomial distribution

$$p(c(\alpha) = r) = b(r; N, p_i) = \binom{N}{r} p^r (1 - p)^{N-r}$$
Derivation of Good-Turing (2)

- Goal of Good-Turing smoothing: compute expected count $c^*$

- Expected count can be computed with help from binomial distribution:

$$E(c^*(\alpha)) = \sum_{r=0}^{N} r \, p(c(\alpha) = r)$$

$$= \sum_{r=0}^{N} r \begin{pmatrix} N \\ r \end{pmatrix} p^r (1 - p)^{N-r}$$

- Note again: $p$ is unknown, we cannot actually compute this
Derivation of Good-Turing (3)

• Definition: expected number of n-grams that occur \( r \) times: \( E_N(N_r) \)

• We have \( s \) different n-grams in corpus
  – let us call them \( \alpha_1, \ldots, \alpha_s \)
  – each occurs with probability \( p_1, \ldots, p_s \), respectively

• Given the previous formulae, we can compute

\[
E_N(N_r) = \sum_{i=1}^{s} p(c(\alpha_i) = r)
\]

\[
= \sum_{i=1}^{s} \binom{N}{r} p_i^r (1 - p_i)^{N-r}
\]

• Note again: \( p_i \) is unknown, we cannot actually compute this
Derivation of Good-Turing (4)

- Reflection
  - we derived a formula to compute $E_N(N_r)$
  - we have $N_r$
  - for small $r$: $E_N(N_r) \simeq N_r$

- Ultimate goal compute expected counts $c^*$, given actual counts $c$

$$E(c^*(\alpha)|c(\alpha) = r)$$
Derivation of Good-Turing (5)

• For a particular n-gram $\alpha$, we know its actual count $r$

• Any of the n-grams $\alpha_i$ may occur $r$ times

• Probability that $\alpha$ is one specific $\alpha_i$

\[
p(\alpha = \alpha_i | c(\alpha) = r) = \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}
\]

• Expected count of this n-gram $\alpha$

\[
E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N p_i p(\alpha = \alpha_i | c(\alpha) = r)
\]
Derivation of Good-Turing (6)

- Combining the last two equations:

\[
E(c^*(\alpha)|c(\alpha) = r) = \sum_{i=1}^{s} N \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}
\]

\[
= \sum_{i=1}^{s} \frac{N \ p_i \ p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}
\]

- We will now transform this equation to derive Good-Turing smoothing
Derivation of Good-Turing (7)

- Repeat:

\[
E(c^*(\alpha)|c(\alpha) = r) = \frac{\sum_{i=1}^{s} N \ p_i \ p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}
\]

- Denominator is our definition of expected counts \( E_N(N_r) \)
Derivation of Good-Turing (8)

• Numerator:

$$\sum_{i=1}^{s} N \ p_i \ p(c(\alpha_i) = r) = \sum_{i=1}^{s} N \ p_i \ \binom{N}{r} p_i^r (1 - p_i)^{N-r}$$

$$= N \ \frac{N!}{N - r! r!} p_i^{r+1} (1 - p_i)^{N-r}$$

$$= N \ \frac{(r + 1) \ N + 1!}{N + 1} \ \frac{N}{N - r! r + 1!} p_i^{r+1} (1 - p_i)^{N-r}$$

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

$$\simeq (r + 1) \ E_{N+1}(N_{r+1})$$
Derivation of Good-Turing (9)

- Using the simplifications of numerator and denominator:

\[ r^* = E(c^*(\alpha)|c(\alpha) = r) \]
\[ = \frac{(r + 1) \frac{E_{N+1}(N_{r+1})}{E_N(N_r)}}{E_N(N_r)} \]
\[ \simeq (r + 1) \frac{N_{r+1}}{N_r} \]

- QED
Back-Off

• In given corpus, we may never observe
  – Scottish beer drinkers
  – Scottish beer eaters

• Both have count 0
  → our smoothing methods will assign them same probability

• Better: backoff to bigrams:
  – beer drinkers
  – beer eaters
Interpolation

- Higher and lower order n-gram models have different strengths and weaknesses
  - high-order n-grams are sensitive to more context, but have sparse counts
  - low-order n-grams consider only very limited context, but have robust counts

- Combine them

\[
p_{I}(w_3|w_1, w_2) = \lambda_1 p_1(w_3) \\
\times \lambda_2 p_2(w_3|w_2) \\
\times \lambda_3 p_3(w_3|w_1, w_2)
\]
Recursive Interpolation

- We can trust some histories $w_{i-n+1}, \ldots, w_{i-1}$ more than others

- Condition interpolation weights on history: $\lambda_{w_{i-n+1}, \ldots, w_{i-1}}$

- Recursive definition of interpolation

\[
p^n_I(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \lambda_{w_{i-n+1}, \ldots, w_{i-1}} p_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) + \\
+ (1 - \lambda_{w_{i-n+1}, \ldots, w_{i-1}}) p^n_{n-1}(w_i|w_{i-n+2}, \ldots, w_{i-1})
\]
Back-Off

- Trust the highest order language model that contains n-gram

\[ p_n^{BO}(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \]
\[
\begin{cases} 
\alpha_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) & \text{if } \text{count}_n(w_{i-n+1}, \ldots, w_i) > 0 \\
\alpha_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) & \text{else}
\end{cases}
\]

- Requires
  - adjusted prediction model \( \alpha_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) \)
  - discounting function \( d_n(w_1, \ldots, w_{n-1}) \)
Back-Off with Good-Turing Smoothing

- Previously, we computed n-gram probabilities based on relative frequency

\[ p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \]

- Good Turing smoothing adjusts counts \( c \) to expected counts \( c^* \)

\[ \text{count}^*(w_1, w_2) \leq \text{count}(w_1, w_2) \]

- We use these expected counts for the prediction model (but \( 0^* \) remains \( 0 \))

\[ \alpha(w_2|w_1) = \frac{\text{count}^*(w_1, w_2)}{\text{count}(w_1)} \]

- This leaves probability mass for the discounting function

\[ d_2(w_1) = 1 - \sum_{w_2} \alpha(w_2|w_1) \]
Diversity of Predicted Words

• Consider the bigram histories *spite* and *constant*
  
  – both occur 993 times in Europarl corpus
  
  – only 9 different words follow *spite*
    almost always followed by *of* (979 times), due to expression *in spite of*
  
  – 415 different words follow *constant*
    most frequent: *and* (42 times), *concern* (27 times), *pressure* (26 times),
    but huge tail of singletons: 268 different words

• More likely to see new bigram that starts with *constant* than *spite*

• Witten-Bell smoothing considers diversity of predicted words
Witten-Bell Smoothing

- Recursive interpolation method

- Number of possible extensions of a history $w_1, ..., w_{n-1}$ in training data

$$N_{1+}(w_1, ..., w_{n-1}, \bullet) = |\{w_n : c(w_1, ..., w_{n-1}, w_n) > 0\}|$$

- Lambda parameters

$$1 - \lambda_{w_1, ..., w_{n-1}} = \frac{N_{1+}(w_1, ..., w_{n-1}, \bullet)}{N_{1+}(w_1, ..., w_{n-1}, \bullet) + \sum_{w_n} c(w_1, ..., w_{n-1}, w_n)}$$
Witten-Bell Smoothing: Examples

Let us apply this to our two examples:

\[
1 - \lambda_{\text{spite}} = \frac{N_{1+}(\text{spite}, \bullet)}{N_{1+}(\text{spite}, \bullet) + \sum_{w_n} c(\text{spite}, w_n)}
\]
\[
= \frac{9}{9 + 993} = 0.00898
\]

\[
1 - \lambda_{\text{constant}} = \frac{N_{1+}(\text{constant}, \bullet)}{N_{1+}(\text{constant}, \bullet) + \sum_{w_n} c(\text{constant}, w_n)}
\]
\[
= \frac{415}{415 + 993} = 0.29474
\]
Diversity of Histories

• Consider the word York
  – fairly frequent word in Europarl corpus, occurs 477 times
  – as frequent as foods, indicates and providers
→ in unigram language model: a respectable probability

• However, it almost always directly follows New (473 times)

• Recall: unigram model only used, if the bigram model inconclusive
  – York unlikely second word in unseen bigram
  – in back-off unigram model, York should have low probability
Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account

- Count of histories for a word

\[ N_{1+}(\bullet w) = |\{w_i : c(w_i, w) > 0\}| \]

- Recall: maximum likelihood estimation of unigram language model

\[ p_{ML}(w) = \frac{c(w)}{\sum_i c(w_i)} \]

- In Kneser-Ney smoothing, replace raw counts with count of histories

\[ p_{KN}(w) = \frac{N_{1+}(\bullet w)}{\sum_{w_i} N_{1+}(w_i \cdot w)} \]
Modified Kneser-Ney Smoothing

- Based on interpolation

\[
p^B_O(w_i|w_{i-n+1}, \ldots, w_{i-1}) =
\begin{cases}
  \alpha_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) \\
  d_n(w_{i-n+1}, \ldots, w_{i-1}) p^B_O(w_i|w_{i-n+2}, \ldots, w_{i-1})
\end{cases}
\]

\text{if } \text{count}_n(w_{i-n+1}, \ldots, w_i) > 0

\text{else}

- Requires
  - adjusted prediction model \( \alpha_n(w_i|w_{i-n+1}, \ldots, w_{i-1}) \)
  - discounting function \( d_n(w_1, \ldots, w_{n-1}) \)
Formula for $\alpha$ for Highest Order N-Gram Model

- Absolute discounting: subtract a fixed $D$ from all non-zero counts

\[
\alpha(w_n|w_1, \ldots, w_{n-1}) = \frac{c(w_1, \ldots, w_n) - D}{\sum_w c(w_1, \ldots, w_{n-1}, w)}
\]

- Refinement: three different discount values

\[
D(c) = \begin{cases} 
D_1 & \text{if } c = 1 \\
D_2 & \text{if } c = 2 \\
D_{3+} & \text{if } c \geq 3
\end{cases}
\]
Discount Parameters

• Optimal discounting parameters $D_1, D_2, D_{3+}$ can be computed quite easily

\[
Y = \frac{N_1}{N_1 + 2N_2}
\]

\[
D_1 = 1 - 2Y \frac{N_2}{N_1}
\]

\[
D_2 = 2 - 3Y \frac{N_3}{N_2}
\]

\[
D_{3+} = 3 - 4Y \frac{N_4}{N_3}
\]

• Values $N_c$ are the counts of n-grams with exactly count $c$
Formula for $d$ for Highest Order N-Gram Model

- Probability mass set aside from seen events

$$d(w_1, ..., w_{n-1}) = \frac{\sum_{i \in \{1,2,3+\}} D_i N_i(w_1, ..., w_{n-1} \bullet)}{\sum_w c(w_1, ..., w_n)}$$

- $N_i$ for $i \in \{1, 2, 3+\}$ are computed based on the count of extensions of a history $w_1, ..., w_{n-1}$ with count 1, 2, and 3 or more, respectively.

- Similar to Witten-Bell smoothing
Formula for $\alpha$ for Lower Order N-Gram Models

- Recall: base on count of histories $N_{1+}(\bullet w)$ in which word may appear, not raw counts.

$$\alpha(w_n|w_1, \ldots, w_{n-1}) = \frac{N_{1+}(\bullet w_1, \ldots, w_n) - D}{\sum_w N_{1+}(\bullet w_1, \ldots, w_{n-1}, w)}$$

- Again, three different values for $D$ ($D_1$, $D_2$, $D_3$), based on the count of the history $w_1, \ldots, w_{n-1}$
Formula for $d$ for Lower Order N-Gram Models

- Probability mass set aside available for the $d$ function

$$d(w_1, ..., w_{n-1}) = \frac{\sum_{i \in \{1,2,3\}} D_i N_i(w_1, ..., w_{n-1})}{\sum_{w_n} c(w_1, ..., w_n)}$$
Interpolated Back-Off

- Back-off models use only highest order n-gram
  - if sparse, not very reliable.
  - two different n-grams with same history occur once $\rightarrow$ same probability
  - one may be an outlier, the other under-represented in training

- To remedy this, always consider the lower-order back-off models

- Adapting the $\alpha$ function into interpolated $\alpha_I$ function by adding back-off

\[
\alpha_I(w_n|w_1, \ldots, w_{n-1}) = \alpha(w_n|w_1, \ldots, w_{n-1}) \\
+ d(w_1, \ldots, w_{n-1}) \cdot p_I(w_n|w_2, \ldots, w_{n-1})
\]

- Note that $d$ function needs to be adapted as well
## Evaluation

Evaluation of smoothing methods:
Perplexity for language models trained on the Europarl corpus

<table>
<thead>
<tr>
<th>Smoothing method</th>
<th>bigram</th>
<th>trigram</th>
<th>4-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good-Turing</td>
<td>96.2</td>
<td>62.9</td>
<td>59.9</td>
</tr>
<tr>
<td>Witten-Bell</td>
<td>97.1</td>
<td>63.8</td>
<td>60.4</td>
</tr>
<tr>
<td>Modified Kneser-Ney</td>
<td>95.4</td>
<td>61.6</td>
<td>58.6</td>
</tr>
<tr>
<td>Interpolated Modified Kneser-Ney</td>
<td>94.5</td>
<td>59.3</td>
<td>54.0</td>
</tr>
</tbody>
</table>
Managing the Size of the Model

• Millions to billions of words are easy to get
  (trillions of English words available on the web)

• But: huge language models do not fit into RAM
Number of Unique N-Grams

Number of unique n-grams in Europarl corpus
29,501,088 tokens (words and punctuation)

<table>
<thead>
<tr>
<th>Order</th>
<th>Unique n-grams</th>
<th>Singletons</th>
</tr>
</thead>
<tbody>
<tr>
<td>unigram</td>
<td>86,700</td>
<td>33,447 (38.6%)</td>
</tr>
<tr>
<td>bigram</td>
<td>1,948,935</td>
<td>1,132,844 (58.1%)</td>
</tr>
<tr>
<td>trigram</td>
<td>8,092,798</td>
<td>6,022,286 (74.4%)</td>
</tr>
<tr>
<td>4-gram</td>
<td>15,303,847</td>
<td>13,081,621 (85.5%)</td>
</tr>
<tr>
<td>5-gram</td>
<td>19,882,175</td>
<td>18,324,577 (92.2%)</td>
</tr>
</tbody>
</table>

→ remove singletons of higher order n-grams
Estimation on Disk

- Language models too large to **build**
- What needs to be stored in RAM?
  - maximum likelihood estimation
    \[
    p(w_n | w_1, ..., w_{n-1}) = \frac{\text{count}(w_1, ..., w_n)}{\text{count}(w_1, ..., w_{n-1})}
    \]
    - can be done separately for each history \( w_1, ..., w_{n-1} \)

- Keep data on disk
  - extract all n-grams into files on-disk
  - sort by history on disk
  - only keep n-grams with shared history in RAM

- Smoothing techniques may require additional statistics
Efficient Data Structures

- Need to store probabilities for
  - the very large majority
  - the very language number

- Both share history the very large

→ no need to store history twice

→ Trie
Fewer Bits to Store Probabilities

- Index for words
  - two bytes allow a vocabulary of $2^{16} = 65,536$ words, typically more needed
  - Huffman coding to use fewer bits for frequent words.

- Probabilities
  - typically stored in log format as floats (4 or 8 bytes)
  - quantization of probabilities to use even less memory, maybe just 4-8 bits
Reducing Vocabulary Size

• For instance: each number is treated as a separate token

• Replace them with a number token NUM
  – but: we want our language model to prefer

\[ p_{LM}(I \text{ pay } 950.00 \text{ in May } 2007) > p_{LM}(I \text{ pay } 2007 \text{ in May } 950.00) \]

  – not possible with number token

\[ p_{LM}(I \text{ pay NUM in May NUM}) = p_{LM}(I \text{ pay NUM in May NUM}) \]

• Replace each digit (with unique symbol, e.g., @ or 5), retain some distinctions

\[ p_{LM}(I \text{ pay 555.55 in May 5555}) > p_{LM}(I \text{ pay 5555 in May 555.55}) \]
Filtering Irrelevant N-Grams

- We use language model in decoding
  - we only produce English words in translation options
  - filter language model down to n-grams containing only those words

- Ratio of 5-grams needed to all 5-grams (by sentence length):
Summary

- Language models: How likely is a string of English words good English?
- N-gram models (Markov assumption)
- Perplexity
- Count smoothing
  - add-one, add-$\alpha$
  - deleted estimation
  - Good Turing
- Interpolation and backoff
  - Good Turing
  - Witten-Bell
  - Kneser-Ney
- Managing the size of the model