

---

# Empirical Methods in Natural Language Processing

## Lecture 4

### Language Modeling (II): Smoothing and Back-Off

Philipp Koehn

17 January 2008



# Language Modeling Example

- Training set

*there is a big house*  
*i buy a house*  
*they buy the new house*

- Model

$p(\text{big} \text{a}) = 0.5$	$p(\text{is} \text{there}) = 1$	$p(\text{buy} \text{they}) = 1$
$p(\text{house} \text{a}) = 0.5$	$p(\text{buy} \text{i}) = 1$	$p(\text{a} \text{buy}) = 0.5$
$p(\text{new} \text{the}) = 1$	$p(\text{house} \text{big}) = 1$	$p(\text{the} \text{buy}) = 0.5$
$p(\text{a} \text{is}) = 1$	$p(\text{house} \text{new}) = 1$	$p(\text{they}   < s >) = .333$

- Test sentence  $S$ : *they buy a big house*

- $p(S) = \underbrace{0.333}_{\text{they}} \times \underbrace{1}_{\text{buy}} \times \underbrace{0.5}_{\text{a}} \times \underbrace{0.5}_{\text{big}} \times \underbrace{1}_{\text{house}} = 0.0833$

## Evaluation of language models

- We want to evaluate the quality of language models
- A good language model gives a high probability to real English
- We measure this with cross entropy and perplexity

## Cross-entropy

- Average entropy of each word prediction

- Example:  $p(S) = \underbrace{0.333}_{\text{they}} \times \underbrace{1}_{\text{buy}} \times \underbrace{0.5}_{\text{a}} \times \underbrace{0.5}_{\text{big}} \times \underbrace{1}_{\text{house}} = 0.0833$

$$\begin{aligned}
 H(p, m) &= -\frac{1}{5} \log p(S) \\
 &= -\frac{1}{5} (\underbrace{\log 0.333}_{\text{they}} + \underbrace{\log 1}_{\text{buy}} + \underbrace{\log 0.5}_{\text{a}} + \underbrace{\log 0.5}_{\text{big}} + \underbrace{\log 1}_{\text{house}}) \\
 &= -\frac{1}{5} (\underbrace{-1.586}_{\text{they}} + \underbrace{0}_{\text{buy}} + \underbrace{-1}_{\text{a}} + \underbrace{-1}_{\text{big}} + \underbrace{0}_{\text{house}}) = 0.7173
 \end{aligned}$$

# Perplexity

- **Perplexity** is defined as

$$\begin{aligned} PP &= 2^{H(p,m)} \\ &= 2^{-\frac{1}{n} \sum_{i=1}^n \log m(w_n|w_1, \dots, w_{n-1})} \end{aligned}$$

- In our example  $H(m, p) = 0.7173 \Rightarrow PP = 1.6441$
- Intuitively, perplexity is the average number of choices at each point (weighted by the model)
- Perplexity is the most common measure to evaluate language models

## Perplexity example

prediction	$p_{\text{LM}}$	$-\log_2 p_{\text{LM}}$
$p_{\text{LM}}(i   \langle /s \rangle \langle s \rangle)$	0.109043	3.197
$p_{\text{LM}}(\textit{would}   \langle s \rangle i)$	0.144482	2.791
$p_{\text{LM}}(\textit{like}   i \textit{ would})$	0.489247	1.031
$p_{\text{LM}}(\textit{to}   \textit{would like})$	0.904727	0.144
$p_{\text{LM}}(\textit{commend}   \textit{like to})$	0.002253	8.794
$p_{\text{LM}}(\textit{the}   \textit{to commend})$	0.471831	1.084
$p_{\text{LM}}(\textit{rapporteur}   \textit{commend the})$	0.147923	2.763
$p_{\text{LM}}(\textit{on}   \textit{the rapporteur})$	0.056315	4.150
$p_{\text{LM}}(\textit{his}   \textit{rapporteur on})$	0.193806	2.367
$p_{\text{LM}}(\textit{work}   \textit{on his})$	0.088528	3.498
$p_{\text{LM}}(.   \textit{his work})$	0.290257	1.785
$p_{\text{LM}}(\langle /s \rangle   \textit{work .})$	0.999990	0.000
average		2.633671

## Perplexity for LM of different order

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

## Recap from last lecture

- If we estimate probabilities solely from counts, we give probability 0 to unseen events (bigrams, trigrams, etc.)
- One attempt to address this was with add-one smoothing.

## Add-one smoothing: results

Church and Gale (1991a) experiment: 22 million words training, 22 million words testing, from same domain (AP news wire), counts of bigrams:

Frequency $r$ in training	Actual frequency in test	Expected frequency in test (add one)
0	0.000027	0.000132
1	0.448	0.000274
2	1.25	0.000411
3	2.24	0.000548
4	3.23	0.000685
5	4.21	0.000822

We overestimate 0-count bigrams ( $0.000132 > 0.000027$ ), but since there are so many, they use up so much probability mass that hardly any is left.

## Deleted estimation: results

- Much better:

Frequency $r$ in training	Actual frequency in test	Expected frequency in test (Good Turing)
0	0.000027	0.000037
1	0.448	0.396
2	1.25	1.24
3	2.24	2.23
4	3.23	3.22
5	4.21	4.22

- Still overestimates unseen bigrams (why?)

## Good-Turing discounting

- Method based on the assumption of binomial distribution of frequencies.
- Translate real counts  $r$  for words with adjusted counts  $r^*$ :

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

$N_r$  is the *count of counts*: number of words with frequency  $r$ .

- The probability mass reserved for unseen events is  $N_1/N$ .
- For large  $r$  (where  $N_{r-1}$  is often 0), so various other methods can be applied (don't adjust counts, curve fitting to linear regression). See Manning+Schütze for details.

## Good-Turing discounting: results

- Almost perfect:

Frequency $r$ in training	Actual frequency in test	Expected frequency in test (Good Turing)
0	0.000027	0.000027
1	0.448	0.446
2	1.25	1.26
3	2.24	2.24
4	3.23	3.24
5	4.21	4.22

## Is smoothing enough?

- If two events (bigrams, trigrams) are both seen with the same frequency, they are given the same probability.

n-gram	count
scottish beer is	0
scottish beer green	0
beer is	45
beer green	0

- If there is not sufficient evidence, we may want to **back off** to lower-order n-grams

## Combining estimators

- We would like to use high-order n-gram language models
- ... but there are many ngrams with count 0.

→ Linear interpolation  $p_{li}$  of estimators  $p_n$  of different order  $n$ :

$$\begin{aligned} p_{li}(w_n | w_{n-2}, w_{n-1}) &= \lambda_1 p_1(w_n) \\ &+ \lambda_2 p_2(w_n | w_{n-1}) \\ &+ \lambda_3 p_3(w_n | w_{n-2}, w_{n-1}) \end{aligned}$$

- $\lambda_1 + \lambda_2 + \lambda_3 = 1$

## Recursive Interpolation

- Interpolation can also be defined recursively

$$p_i(w_n | w_{n-2}, w_{n-1}) = \lambda(w_{n-2}, w_{n-1}) p(w_n | w_{n-2}, w_{n-1}) + (1 - \lambda(w_{n-2}, w_{n-1})) p_i(w_n | w_{n-1})$$

- How do we set the  $\lambda(w_{n-2}, w_{n-1})$  parameters?
  - consider  $count(w_{n-2}, w_{n-1})$
  - for higher counts of history:
    - higher values of  $\lambda(w_{n-2}, w_{n-1})$
    - less probability mass reserved for unseen events

## Witten-Bell Smoothing

- Count of history may not be fully adequate
  - **constant** occurs 993 in Europarl corpus, 415 different words follow
  - **spite** occurs 993 in Europarl corpus, 9 different words follow
- Witten-Bell smoothing uses diversity of history
- Reserved probability for unseen events:
  - $1 - \lambda(\text{constant}) = \frac{415}{415+993} = 0.295$
  - $1 - \lambda(\text{spite}) = \frac{9}{9+993} = 0.009$

## Back-off

- Another approach is to back-off to lower order n-gram language models

$$p_{bo}(w_n|w_{n-2}, w_{n-1}) = \begin{cases} \alpha(w_n|w_{n-2}, w_{n-1}) & \text{if } count(w_{n-2}, w_{n-1}, w_n) > 0 \\ \gamma(w_{n-2}, w_{n-1}) p_{bo}(w_n|w_{n-1}) & \text{otherwise} \end{cases}$$

- Each trigram probability distribution is changed to a function  $\alpha$  that reserves some probability mass for unseen events:  $\sum_w \alpha(w_n|w_{n-2}, w_{n-1}) < 1$
- The remaining probability mass is used in the weight  $\gamma(w_{n-2}, w_{n-1})$ , which is given to the back-off path.

## Back-off with Good Turing Discounting

- Good Turing discounting is used for all positive counts

	count	$p$	GT count	$\alpha$
$p(\text{big} \text{a})$	3	$\frac{3}{7} = 0.43$	2.24	$\frac{2.24}{7} = 0.32$
$p(\text{house} \text{a})$	3	$\frac{3}{7} = 0.43$	2.24	$\frac{2.24}{7} = 0.32$
$p(\text{new} \text{a})$	1	$\frac{1}{7} = 0.14$	0.446	$\frac{0.446}{7} = 0.06$

- $1 - (0.32 + 0.32 + 0.06) = 0.30$  is left for back-off  $\gamma(\text{a})$
- Note: actual value for  $\gamma$  is slightly higher, since the predictions of the lower-order model to seen events at this level are not used.

# Absolute Discounting

- Subtract a fixed number  $D$  from each count

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

- Typical counts 1 and 2 are treated differently

## Consider Diversity of Histories

- Words differ in the number of different history they follow
  - **foods**, **indicates**, **providers** occur 447 times each in Europarl
  - **york** also occurs 447 times in Europarl
  - but: **york** almost always follows **new**
- When building a unigram model for back-off
  - what is a good value for  $p(\text{foods})$  ?
  - what is a good value for  $p(\text{york})$  ?

# Kneser-Ney Smoothing

- Currently most popular smoothing method
- Combines
  - absolute discounting
  - considers diversity of predicted words for back-off
  - considers diversity of histories for lower order n-gram models
  - interpolated version: always add in back-off probabilities

## Perplexity for different language models

- Trained on English Europarl corpus, ignoring trigram and 4-gram singletons

<b>Smoothing method</b>	<b>bigram</b>	<b>trigram</b>	<b>4-gram</b>
Good-Turing	96.2	62.9	59.9
Witten-Bell	97.1	63.8	60.4
Modified Kneser-Ney	95.4	61.6	58.6
Interpolated Modified Kneser-Ney	94.5	59.3	54.0

## Other methods in language modeling

- Language modeling is still an active field of research
- There are many back-off and interpolation methods
- Skip n-gram models: back-off to  $p(w_n|w_{n-2})$
- Factored language models: back-off to word stems, part-of-speech tags
- Syntactic language models: using parse trees
- Language models trained on billions and trillions of words