
Language Models

Philipp Koehn

5 September 2023



- **Language models** answer the question:

How likely is a string of English words good English?

- Help with reordering

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

- Help with word choice

$$p_{\text{LM}}(\text{I am going home}) > p_{\text{LM}}(\text{I am going house})$$

N-Gram Language Models



2

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

→ Decomposing $p(W)$ using the chain rule:

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

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

- **Markov assumption:**
 - only previous history matters
 - limited memory: only last k words are included in history (older words less relevant)
- **k th order Markov model**
- For instance 2-gram language model:

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

- What is conditioned on, here w_{i-1} is called the **history**

Estimating N-Gram Probabilities



4

- 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

the green (total: 1748)			the red (total: 225)			the blue (total: 54)		
word	c.	prob.	word	c.	prob.	word	c.	prob.
paper	801	0.458	cross	123	0.547	box	16	0.296
group	640	0.367	tape	31	0.138	.	6	0.111
light	110	0.063	army	9	0.040	flag	6	0.111
party	27	0.015	card	7	0.031	,	3	0.056
ecu	21	0.012	,	5	0.022	angel	3	0.056

- 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:

$$H(W) = -\frac{1}{n} \log_2 p(W_1^n)$$

- Or, **perplexity**

$$\text{perplexity}(W) = 2^{H(W)}$$

Example: 3-Gram

prediction	p_{LM}	$-\log_2 p_{LM}$
$p_{LM}(i </s><s>)$	0.109	3.197
$p_{LM}(\text{would} <s>i)$	0.144	2.791
$p_{LM}(\text{like} i \text{ would})$	0.489	1.031
$p_{LM}(\text{to} \text{would like})$	0.905	0.144
$p_{LM}(\text{commend} \text{like to})$	0.002	8.794
$p_{LM}(\text{the} \text{to commend})$	0.472	1.084
$p_{LM}(\text{rapporteur} \text{commend the})$	0.147	2.763
$p_{LM}(\text{on} \text{the rapporteur})$	0.056	4.150
$p_{LM}(\text{his} \text{rapporteur on})$	0.194	2.367
$p_{LM}(\text{work} \text{on his})$	0.089	3.498
$p_{LM}(.\text{his work})$	0.290	1.785
$p_{LM}(</s> \text{work .})$	0.99999	0.000014
average		2.634

Comparison 1–4-Gram

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	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

count smoothing

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

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

- 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- α Smoothing

- Add $\alpha < 1$ to each count

$$p = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α ?
- Could be optimized on held-out set

What is the Right Count?

- Example:
 - the 2-gram **red circle** occurs in a 30 million word corpus exactly once
 - maximum likelihood estimation tells us that its probability is $\frac{1}{30,000,000}$
 - ... but we would expect it to occur less often than that
- Question: How likely does a 2-gram that occurs once in a 30,000,000 word corpus occur in the wild?
- Let's find out:
 - get the set of all 2-grams that occur once (**red circle**, **funny elephant**, ...)
 - record the size of this set: N_1
 - get another 30,000,000 word corpus
 - for each word in the set: count how often it occurs in the new corpus (many occur never, some once, fewer twice, even fewer 3 times, ...)
 - sum up all these counts (0 + 0 + 1 + 0 + 2 + 1 + 0 + ...)
 - divide by N_1 → that is our test count t_c

Example: 2-Grams in Europarl

Count	Adjusted count		Test count
c	$(c + 1)\frac{n}{n+v^2}$	$(c + \alpha)\frac{n}{n+\alpha v^2}$	t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

- Add- α smoothing with $\alpha = 0.00017$

Deleted Estimation

- Estimate true counts in held-out data
 - split corpus in two halves: training and held-out
 - counts in training $C_t(w_1, \dots, w_n)$
 - 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:

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

- Both halves can be switched and results combined

$$p_h(w_1, \dots, w_n) = \frac{T_r^1 + T_r^2}{N(N_r^1 + N_r^2)} \text{ where } \text{count}(w_1, \dots, w_n) = r$$

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
- Where does this formula come from? Derivation is in the textbook.

Good-Turing for 2-Grams in Europarl

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

adjusted count fairly accurate when compared against the test count

backoff and interpolation

- 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

- 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

$$\begin{aligned} p_I(w_3|w_1, w_2) = & \lambda_1 p_1(w_3) \\ & + \lambda_2 p_2(w_3|w_2) \\ & + \lambda_3 p_3(w_3|w_1, w_2) \end{aligned}$$

Recursive Interpolation

- We can trust some histories $w_{i-n+1}, \dots, w_{i-1}$ more than others
- Condition interpolation weights on history: $\lambda_{w_{i-n+1}, \dots, w_{i-1}}$
- Recursive definition of interpolation

$$\begin{aligned} p_n^I(w_i | w_{i-n+1}, \dots, w_{i-1}) &= \lambda_{w_{i-n+1}, \dots, w_{i-1}} p_n(w_i | w_{i-n+1}, \dots, w_{i-1}) + \\ &+ (1 - \lambda_{w_{i-n+1}, \dots, w_{i-1}}) p_{n-1}^I(w_i | w_{i-n+2}, \dots, w_{i-1}) \end{aligned}$$

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

$$p_n^{BO}(w_i | w_{i-n+1}, \dots, w_{i-1}) = \begin{cases} \alpha_n(w_i | w_{i-n+1}, \dots, w_{i-1}) & \text{if } \text{count}_n(w_{i-n+1}, \dots, w_i) > 0 \\ d_n(w_{i-n+1}, \dots, w_{i-1}) p_{n-1}^{BO}(w_i | w_{i-n+2}, \dots, w_{i-1}) & \text{else} \end{cases}$$

- Requires
 - adjusted prediction model $\alpha_n(w_i | w_{i-n+1}, \dots, w_{i-1})$
 - discounting function $d_n(w_1, \dots, w_{n-1})$

Back-Off with Good-Turing Smoothing

23



- 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

$$d_2(w_1) = 1 - \sum_{w_2} \alpha(w_2|w_1)$$

Example

- Good Turing discounting is used for all positive counts

	count	p	GT count	α
$p(\text{big} \text{a})$	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 $d_2(\text{a})$
- Note: actual values for d_2 is slightly higher, since the predictions of the lower-order model to seen events at this level are not used.

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, \dots, w_{n-1} in training data

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

- Lambda parameters

$$1 - \lambda_{w_1, \dots, w_{n-1}} = \frac{N_{1+}(w_1, \dots, w_{n-1}, \bullet)}{N_{1+}(w_1, \dots, w_{n-1}, \bullet) + \sum_{w_n} c(w_1, \dots, w_{n-1}, w_n)}$$

Witten-Bell Smoothing: Examples

Let us apply this to our two examples:

$$\begin{aligned} 1 - \lambda_{spite} &= \frac{N_{1+}(spite, \bullet)}{N_{1+}(spite, \bullet) + \sum_{w_n} c(spite, w_n)} \\ &= \frac{9}{9 + 993} = 0.00898 \end{aligned}$$

$$\begin{aligned} 1 - \lambda_{constant} &= \frac{N_{1+}(constant, \bullet)}{N_{1+}(constant, \bullet) + \sum_{w_n} c(constant, w_n)} \\ &= \frac{415}{415 + 993} = 0.29474 \end{aligned}$$

- 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

$$p_{KN}(w) = \frac{N_{1+}(\bullet w)}{\sum_{w_i} N_{1+}(\bullet w_i)}$$

Modified Kneser-Ney Smoothing

- Based on interpolation

$$p_n^{BO}(w_i | w_{i-n+1}, \dots, w_{i-1}) =$$
$$= \begin{cases} \alpha_n(w_i | w_{i-n+1}, \dots, w_{i-1}) & \text{if } \text{count}_n(w_{i-n+1}, \dots, w_i) > 0 \\ d_n(w_{i-n+1}, \dots, w_{i-1}) p_{n-1}^{BO}(w_i | w_{i-n+2}, \dots, w_{i-1}) & \text{else} \end{cases}$$

- Requires
 - adjusted prediction model $\alpha_n(w_i | w_{i-n+1}, \dots, w_{i-1})$
 - discounting function $d_n(w_1, \dots, w_{n-1})$

Formula for α for Highest Order N-Gram Model

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

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

- Refinement: three different discount values

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

- 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

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

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

Formula for α for Lower Order N-Gram Models



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

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

- Again, three different values for D (D_1 , D_2 , D_{3+}), based on the count of the history w_1, \dots, w_{n-1}

Formula for d for Lower Order N-Gram Models



- Probability mass set aside available for the d function

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

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 → 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 α function into interpolated α_I function by adding back-off

$$\begin{aligned}\alpha_I(w_n|w_1, \dots, w_{n-1}) &= \alpha(w_n|w_1, \dots, w_{n-1}) \\ &\quad + d(w_1, \dots, w_{n-1}) p_I(w_n|w_2, \dots, w_{n-1})\end{aligned}$$

- Note that d function needs to be adapted as well

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

Smoothing method	bigram	trigram	4-gram
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

efficiency

Managing the Size of the Model

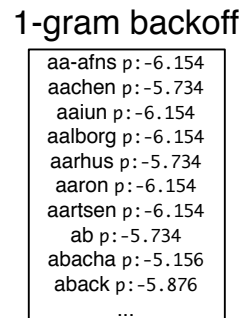
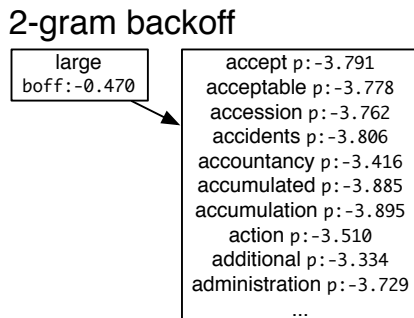
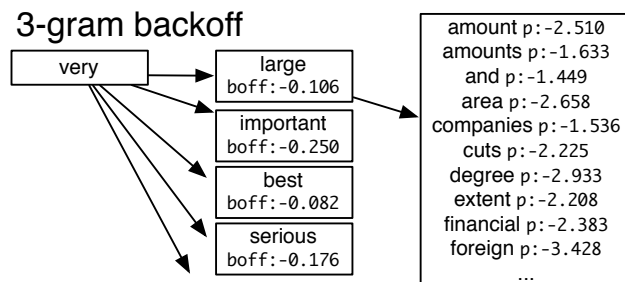
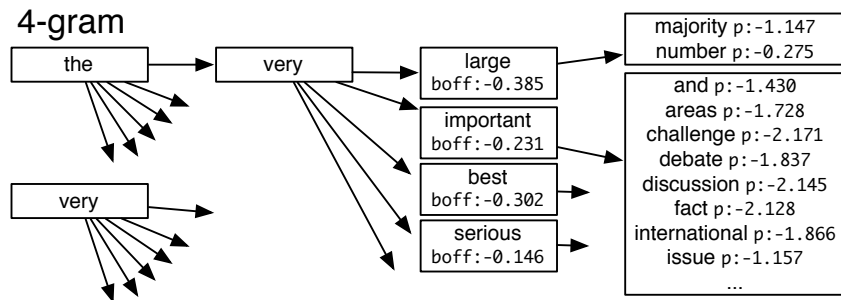
- Millions to billions of words are easy to get
(trillions of English words available on the web)
- But: huge language models do not fit into RAM

Number of Unique N-Grams

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

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

→ remove singletons of higher order n-grams



- Need to store probabilities for
 - the very large majority
 - the very large number
 - Both share history
 - the very large
- no need to store history twice
- Trie

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

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

- not possible with number token

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

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

$$p_{\text{LM}}(\text{I pay 555.55 in May 5555}) > p_{\text{LM}}(\text{I pay 5555 in May 555.55})$$

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