N-gram Language Models [1]

Erhard Hinrichs

Seminar für Sprachwissenschaft
Eberhard-Karls Universität Tübingen

[1] Largely based on material from Jurafsky and Martin's textbook *Speech and Language Processing*, Chapter 3. available at:
`https://web.stanford.edu/~jurafsky/slp3/`

What are Language Models (LMs)?

- LMs assign probabilities to sequences of characters or words of a language: $P(w_1, w_2, w_3, w_4, \ldots, w_n)$
- LMs predict the likelihood of the next character or word in a sequence: $P(w_n \mid w_1, w_2, w_3, \ldots, w_{n-1})$
- Example:
  - **sq** $\rightarrow$ **squ** $\rightarrow$ **squa** $\rightarrow$ **squar** $\rightarrow$ **square**
  - **sq** $\rightarrow$ **sqr** $\rightarrow$ **sqrt**
  - **Can please come** $\rightarrow$ **here?**
  - **Can you please go** $\rightarrow$ **there?**

Query continuations

what is the weather

- what is the weather
- what is the weather **today**
- what is the weather **like**
- what is the weather **like today**
- what is the weather **like auf deutsch**
- what is the weather **for tomorrow**
- what is the weather **now**
- what is the weather **like tomorrow**
- what is the weather **going to be tomorrow**
- what is the weather **in germany**

NLP Applications of Language Modelling

- Autocompletion of words
- Automatic speech recognition
- Automatic spelling correction
- Automatic grammar checking and correction
- Machine translation
- Augmentative and alternative communication (AAC)
- Part of speech tagging
- Chatbots
- Generative AI; e.g. Chat-GPT

How can probabilities of sequences be computed?

- Using the chain rule of probabilities
- Approximation of the context history of a word by n-grams (e.g. bigrams, trigrams, etc.)

Some notational conventions

- $P(X_1 = \text{"}can\text{"})$, where P() is a probability and X_1 a random variable.
  - often abbreviated as: $P(\text{"}can\text{"})$
- a sequence of N words: $w_1 \ldots w_n$
  - often abbreviated as: $w_{1:n}$
  - special cases:

    $w_{1:1} = w_1$

    $w_{1:0} =$ a sequence of length $0 =$ no sequence

Decomposing the joint probability of a sequence by the chain rule of probability

$$
\begin{aligned}
P(X_1 \dots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \dots P(X_n|P(X_n|X_{1:n-1}) \\
&= \prod_{k=1}^{n} P(X_k|X_{1:k-1})
\end{aligned}
\tag{3.3}
$$

$$
\begin{aligned}
P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \dots P(w_n|w_{1:n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_{1:k-1})
\end{aligned}
\tag{3.4}
$$

N-gram approximation

Approximation of conditional probability of the next word for a bigram model:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1}) \tag{3.7}$$

For N-gram size N, approximation of a word w_n given its entire context $w_{1:n-1}$:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1}) \tag{3.8}$$

Bigram approximation of the probability of word sequence $w_{1:n}$:

$$P(w_{1:n}) = \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{3.9}$$

Maximum likelihood estimation

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \qquad (3.10 - 3.11)$$

Comment: The sum of all bigram counts that start with a given word w_{n-1} is equal to the unigram count of that word w_{n-1}

MLE n-gram Parameter Estimation

$$P(w_n \mid w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})} \tag{3.12}$$

Example: Let N = 3 and n = 3

$$P(w_3 \mid w_{1:2}) = \frac{C(w_{1:2}w_3)}{C(w_{1:2})}$$

Example 1: Mini Corpus of 3 Sentences

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Calculations for some bigram probabilities from this corpus

$$P(I \mid <s>) = \frac{2}{3} \qquad P(Sam \mid <s>) = \frac{1}{3} \qquad P(am \mid I) = \frac{2}{3}$$
$$P(</s> \mid Sam) = \frac{1}{2} \qquad P(Sam \mid am) = \frac{1}{2} \qquad P(do \mid I) = \frac{1}{3}$$

We need the end-symbol to make the bigram grammar a true probability distribution. Without an end-symbol, the sentence probabilities for all sentences of a given length would sum to one. This model would define an infinite set of probability distributions, with one distribution per sentence length.

SLP Exercise 3.5: Why we need the end-symbol?

Suppose we didn't use the end-symbol </s>. Train an unsmoothed bigram grammar on the following training corpus without using the end-symbol </s>:

<s> a b
<s> b b
<s> b a
<s> a a

Demonstrate that your bigram model does not assign a single probability distribution across all sentence lengths by showing that the sum of the probability of the four possible 2 word sentences over the alphabet $\{a, b\}$ is 1.0, and the sum of the probability of all possible 3 word sentences over the alphabet $\{a, b\}$ is also 1.0.

Solution

The last word in each sentence in the training data is never counted as the beginning of a bigram. When we have end-of-sentence markers, we don't lose information because nothing ever comes after the end-of-sentence marker. When we remove the end-of-sentence markers, we lose information about the last word of each sentence of our training data. The bigrams are:

<s> a b
<s> b b
<s> b a
<s> a a

counts	<s>	a	b	sum
<s>	0	2	2	4
a	0	1	1	2
b	0	1	1	2

prob.	<s>	a	b
<s>	0	1/2	1/2
a	0	1/2	1/2
b	0	1/2	1/2

Solution (continued)

So the probabilities of all possible 2-word sentences are:

$$P(\text{<s> } aa) = P(a|\text{<s>})P(a|a) = 1/2 \times 1/2 = 1/4$$
$$P(\text{<s> } ab) = P(a|\text{<s>})P(b|a) = 1/2 \times 1/2 = 1/4$$
$$P(\text{<s> } ba) = P(b|\text{<s>})P(a|b) = 1/2 \times 1/2 = 1/4$$
$$P(\text{<s> } bb) = P(b|\text{<s>})P(b|b) = 1/2 \times 1/2 = 1/4$$

Solution (continued)

And the probabilities of all possible 3-word sentences are:

$$P(\text{<s> } aaa) = P(a|\text{<s>})P(a|a)P(a|a) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } aab) = P(a|\text{<s>})P(a|a)P(b|a) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } aba) = P(a|\text{<s>})P(b|a)P(a|b) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } abb) = P(a|\text{<s>})P(b|a)P(b|b) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } baa) = P(b|\text{<s>})P(a|b)P(a|a) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } bab) = P(b|\text{<s>})P(a|b)P(b|a) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } bba) = P(b|\text{<s>})P(b|b)P(a|b) = 1/2 \times 1/2 \times 1/2 = 1/8$$
$$P(\text{<s> } bbb) = P(b|\text{<s>})P(b|b)P(b|b) = 1/2 \times 1/2 \times 1/2 = 1/8$$

Calculation with end-symbol

<s> a b </s>
<s> b b </s>
<s> b a </s>
<s> a a </s>

cnts	<s>	a	b	</s>	sum
<s>	0	2	2	0	4
a	0	1	1	2	4
b	0	1	1	2	4

pr.	<s>	a	b	</s>
<s>	0	1/2	1/2	0
a	0	1/4	1/4	1/2
b	0	1/4	1/4	1/2

The probabilities of all possible 2-word sentences are:

$P(\text{<s>aa</s>}) = P(a|\text{<s>})P(a|a)P(\text{</s>}|\text{a}) = 1/2 \times 1/4 \times 1/2 = 1/16$
$P(\text{<s>ab</s>}) = P(a|\text{<s>})P(b|a)P(\text{</s>}|\text{b}) = 1/2 \times 1/4 \times 1/2 = 1/16$
$P(\text{<s> } ba\text{</s>}) = P(b|\text{<s>})P(a|b)P(\text{</s>}|\text{a}) = 1/2 \times 1/4 \times 1/2 = 1/16$
$P(\text{<s> } bb\text{</s>}) = P(b|\text{<s>})P(b|b)P(\text{</s>}|\text{b}) = 1/2 \times 1/4 \times 1/2 = 1/16$

Calculation with end-symbol

The probabilities of all possible 3-word sentences are:

$$
\begin{aligned}
P(\text{<s> } aaa \text{</s>}) &= P(a|\text{<s>})P(a|a)P(a|a)P(\text{</s>}|a) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } aab \text{</s>}) &= P(a|\text{<s>})P(a|a)P(b|a)P(\text{</s>}|b) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } aba \text{</s>}) &= P(a|\text{<s>})P(b|a)P(a|b)P(\text{</s>}|a) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } abb \text{</s>}) &= P(a|\text{<s>})P(b|a)P(b|b)P(\text{</s>}|b) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } baa \text{</s>}) &= P(b|\text{<s>})P(a|a)P(a|a)P(\text{</s>}|a) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } bab \text{</s>}) &= P(b|\text{<s>})P(a|b)P(b|a)P(\text{</s>}|b) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } bba \text{</s>}) &= P(b|\text{<s>})P(b|b)P(a|b)P(\text{</s>}|a) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \\
P(\text{<s> } bbb \text{</s>}) &= P(b|\text{<s>})P(b|b)P(b|b)P(\text{</s>}|b) \\
&= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64
\end{aligned}
$$

Example 2: Bigram Model for Berkeley Restaurant Corpus

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Example 2: Bigram Model for Berkeley Restaurant Corpus

Unigram Counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Some bigram calculations:

$$P(i, want) = \frac{C(i, want)}{C(i)} = \frac{827}{2533} \approx 0.3265 \quad (1)$$

$$P(want, to) = \frac{C(want, to)}{C(want)} = \frac{608}{927} \approx 0.6558 \quad (2)$$

Why Use log Probabilities

$$p_1 \times p_2 \times p_3 \times p_4 = Exp(logp_1 + logp_2 + logp_3 + logp_4) \quad (3)$$

- Adding in log space is equivalent multiplying in linear space.
- Prevents underflow: the more probabilities we multiply together, the smaller the product becomes.

Evaluating Language Models: Intrinsic Evaluation

Dividing the data up in:

- **Training set**: the data to train the model with
- **Development set**: the data used to fine-tune the model with
- **Test set**: previously unseen data to test the model with

Evaluation of NLP Models

- Every natural language model needs to be properly evaluated. This is often a non-trivial task that ideally should be supported by the following **FAIR**[2] criteria:
- **Findable**: good metadata for data used and for NLP model
- **Accessible**: Open access to the source code of the model and the data used for training the model
- **Interoperable**: Integrating data and applications into workflows with other resources
- **Re-usable**: (Meta)data meet community standards for data encoding and include license information

[2] see `https://www.go-fair.org/fair-principles/`

Extrinsic and Intrinsic Evaluation

- Extrinsic evaluation (also know as end-to-end evaluation) evaluates a model in the context of a particular task or sequence (pipeline of tasks).
  - Example: evaluating a sentence-splitting tool in the context of a tokenizer or a lemmatizer or a part-of-speech tagger in the context of a parser.
  - Example: evaluating the quality of a speech recognition system in a car by the average number of clarifications necessary to identify the correct destination.
- Intrinsic evaluation: follows a standardized testing regime, ideally using shared data sets.

Intrinsic Evaluation: Dividing the data up in:

- **Training set**: the data to train the model with
- **Development set**: the data used to fine-tune the model with
- **Test set**: previously unseen data to test the model with

An absolute no no: training on the test data!

Perplexity of a Corpus

The perplexity of a test set according to a language model is the geometric mean of the inverse test set probability computed by the model.

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\
&= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}
\end{aligned}
\qquad (3.14 - 3.16)
$$

where:

- N is the word count for the entire corpus.

Additional Assumptions

- Each sentence is flanked with a *begin sentence* tag $< s >$ and an *end sentence* tag $< /s >$. Both tags need to be included in the probability computation.
- $< /s >$ (but not $< s >$) needs to be included in the set N of word tokens.
- See discussion of n-gram computation and ensuring proper probability distribution over any sentence lengths

Perplexity and Information Theory

- Perplexity as the weighted average branching factor.
- The branching factor of a language is the number of possible next words that can follow any word.
- Perplexity is closely related to the notion of entropy in information theory.

Example: Mini-language of digits with the task of digit recognition

Scenario 1:

- Each of the digits in English (zero, ... nine) occurs with equal probability $P = \frac{1}{10}$.
- The resulting perplexity is 10.

Imagine a test string of digits of length 10 and assume that in the training set all the digits occurred with equal probability.

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{\frac{1}{N}} \\ &= \left(\left(\frac{1}{10}\right)^N\right)^{-\frac{1}{N}} = \left(\frac{1}{10}\right)^{-1} = 10 \end{aligned} \tag{3.17}$$

Example: Mini-language of digits with the task of digit recognition

Scenario 2:
You are given a training set of 100 numbers that consists of 91 zeros and 1 each of the other digits 1-9. Now we see the following test set: 0 0 0 0 0 3 0 0 0 0. What is the unigram perplexity?

Solution for Scenario 2

Scenario 2:

$$PP(0000030000) = P(0000030000)^{\frac{1}{10}}$$
$$= \left(\left(\frac{91}{100}\right)^{9} \times \frac{1}{100}\right)^{-\frac{1}{10}} \approx .580$$

Entropy

Entropy is the measure of uncertainty of a random variable. (measured in bits). The entropy of a discrete random variable X

$$H(X) = \sum_{x \in X} p(x) log_2(1/p(x)) \tag{3.41}$$

Entropy of a fair coin:

$$H(P) = 1/2 \times log(2) + 1/2 \times log(2) = 1/2 + 1/2 = 1$$

Example: Comparison of different n-gram Models

In general: The more information the n-gram gives us about the sequence, the lower the perplexity.

Scenario:

- unigram, bigram, and trigram models trained on 38 million words (including start-of-sentence tokens) from the *Wall Street Journal*, using a 19,979 word vocabulary.
- Test set: 1.5 million words WSJ for each of these trained n-gram models

Results:

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Final Remarks on Perplexity

- The perplexity of two language models is only comparable of they use identical vocabularies.
- An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition or machine translation.
- A model's improvement in perplexity should always be confirmed by an end-to-end evaluation of a real task before concluding the evaluation of the model.

Visualizing Distributions and Sampling

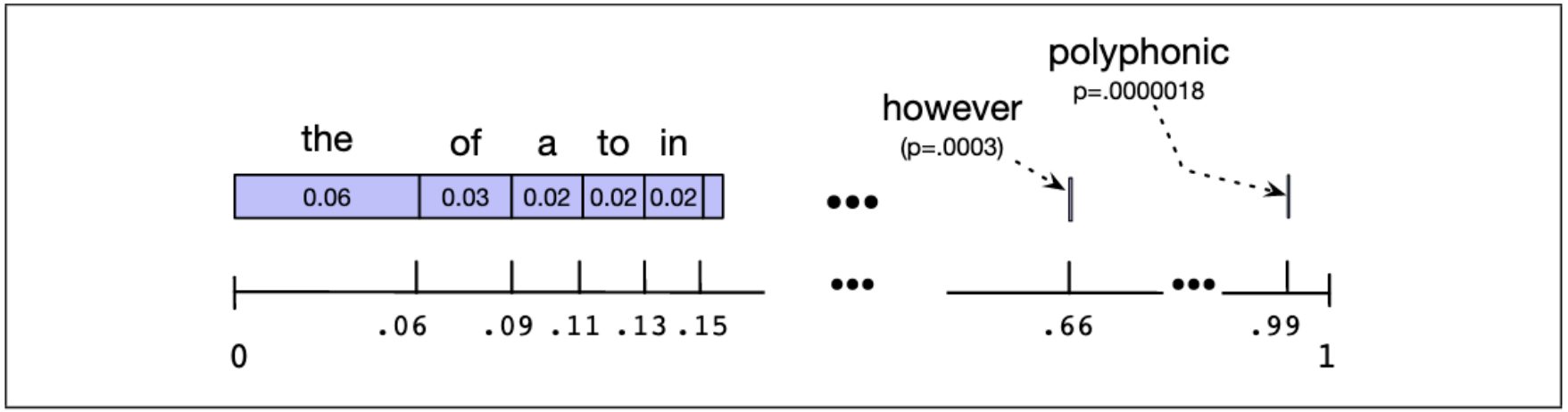

Figure 3.3 A visualization of the sampling distribution for sampling sentences by repeatedly sampling unigrams. The blue bar represents the frequency of each word. The number line shows the cumulative probabilities. If we choose a random number between 0 and 1, it will fall in an interval corresponding to some word. The expectation for the random number to fall in the larger intervals of one of the frequent words (*the*, *of*, *a*) is much higher than in the smaller interval of one of the rare words (*polyphonic*).

N-grams from Wall Street Journal Data

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 3.5 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

N-grams from Shakespeare Data

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
	–Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
	–What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
	–This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
	–It cannot be but so.

Figure 3.4 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

Some general tendencies about n-gram language models

- The longer the n-gram, the more coherent a sampled text will appear.
  - Unigram models provide no context windows and therefore no word-to-word relationships
  - Bigram models provide "local" information about word-to-word dependencies within bigrams.
  - Trigram and higher values of N lead to more fluent text.
- There is a trade-off between the context size of an N-gram model and the sparsity of the N-grams.
- The training corpus is dependent on the subject domain or the genre as well as the mix of training data.

Data Sparseness and How to Deal with it

- Missing n-grams in the training data:
  - Accidental gaps in coverage for trigrams such as *denied the loan* or *denied the offer*, which may appear in the test data, but not in the training data
  - Such zero counts are a problem because their presence is underestimated
  - If the probability of word is 0, then the entire probability of the test set is zero.

Unknown Words: Option 1

1. **Choose** a vocabulary (word list) that is fixed in advance. This is possible if the word list of the application domain is finite and of managable size.
2. **Convert** in the training set any word that is not in this set (any OOV word) to the unknown word token <UNK> in a text normalization step.
3. **Estimate** the probabilities for <UNK> from its counts just like any other regular word in the training set.

Unknown Words: Option 2

1. Do not choose a vocabulary that is fixed in advance.
2. Replace by <UNK> all words based on their frequency. Replace by <UNK> all words that occur fewer than n times, where n is some small number.
3. or Equivalently select a vocabulary size V in advance (say 50,000) and choose the top V words by frequency and replace the rest by UNK

How to Overcome Data Sparseness

- Unseen events have zero probability
- Solution: redistributing probability mass from seen events to unseen events (smoothing, discounting)

Smoothing Methods – Overview

- Laplace smoothing, add-one smoothing
- Add-k smoothing
- Backoff and Interpolation
- Kneser-Ney smoothing

Laplace Smoothing

- Also known as *add-one smoothing*
- adds a count of 1 to all n-grams.
- Laplace smoothing for unigrams probabilities:

$$P(w_i) = \frac{c_i}{N} \qquad \text{(unsmoothed unigram probabilities)}$$

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} \tag{3.20}$$

Smoothed Unigram Counts

$$c_i^* = (c_i + 1)\frac{N}{N + V} \tag{3.21}$$

Instead of referring to the discounted counts c^*, the relative **discount** d_c, the ratio between the discounted counts and the original count, can be computed as follows:

$$d_c = \frac{c^*}{c}$$

This reveals the probability mass re-assigned from non-zero counts to the zero counts.

Add-one Smoothed Bigram Counts for Words from BeRP Corpus

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 3.6 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

Unsmoothed Bigrams versus Laplace Smoothed Bigrams

Unsmoothed Bigram MLE (formula repeated from above):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \qquad (3.10 - 3.11)$$

For add-one smoothed bigram counts, we need to augment the unigram count by the number of total word types in the vocabulary V:

$$P^*_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \qquad (3.22)$$

Thus, each of the unigram counts given in the previous section will need to be augmented by V = 1446.

Add-one Smoothed Bigram Probabilities for Words from BeRP Corpus

	i	want	to	eat	chinese	food	lunch
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058

Figure 3.7 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in corpus of 9332 sentences. Previously-zero probabilities are in gray.

Adjusted Counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V} \tag{3.24}$$

Add-one Re-constituted Counts

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figure 3.8 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero counts are in gray.

Laplace, Add-k, and Unigram Prior Smoothing

$$P^*_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \tag{3.22}$$

$$P^*_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \tag{3.25}$$

$$P^*_{UnigramPrior}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k(P(w_n))}{C(w_{n-1}) + k}$$

Backoff and Interpolation

General strategy:

- Sometimes using less context is a good strategy, helping to generalize more for contexts that the model hasn't learnt much about.

Two methods:

- linear interpolation: combines different order n-grams, each weighted by a parameter λ
- backoff: backing-off to a lower order n-gram if we have zero evidence for a higher-order n-gram. λ can be learnt by optimizing the likelihood of the development set.

Backoff and Interpolation

Simple linear interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = &\lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+ \lambda_2 P(w_n|w_{n-1}) \\ &+ \lambda_3 P(w_n)\end{aligned} \tag{3.26}$$

Interpolation with context-conditioned weights:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = &\lambda_1(w_{n-2:n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &+ \lambda_2(w_{n-2:n-1}) P(w_n|w_{n-1}) \\ &+ \lambda_3(w_{n-2:n-1}) P(w_n)\end{aligned} \tag{3.28}$$

This allows for giving more or less weights to n-grams that are more or less reliable than the average.

Examples of high-frequency trigrams in the COCA corpus

Top ten most frequent continuations of the bigram *slice of* in the COCA corpus:
bread (1510), *cheese* (667), *pie* (567), *pizza* (556), *tomato* (530), *bacon* (465), *lemon* (453), *apple* (435), *cake* (379), *onion* (334)

The Corpus of Contemporary American English (COCA)

- the only large and "representative" corpus of American English
- current size: more than one billion words of text (25+ million words each year 1990-2019)
- combination of eight genres: spoken, fiction, popular magazines, newspapers, academic texts, and TV and Movies subtitles, blogs, and other web pages
- `https://www.english-corpora.org/coca/`

Katz backoff

In Katz backoff we rely on a discounted probability P^* if we have seen this n-gram before (i.e., if we have non-zero counts).

$$P_{BO}(w_n|w_{n-N+1:n-1}) =$$

$$\begin{cases} P^*(w_n|w_{n-N+1:n-1}) & \text{if } C(w_{n-N+1:n-1}) > 0 \\ \alpha(w_{n-N+1:n-1})P_{BO}(w_n|w_{n-N+2:n-1}) & \text{otherwise} \end{cases} \tag{3.29}$$

The parameter α is needed for redistributing probability mass from higher order n-grams to lower order n-grams so as to maintain a proper probability distribution. The value of α can be learnt from maximizing the likelihood of the development set.

Underlying Intuition (due to I.J. Good and Alan Turing) for More Advanced Smoothing Algorithms

- Use the count of things you have **seen once** to help estimate the things you have **never seen up until now**
- In order to operationalize this intuition, we need to introduce the notion of N_c which refers to the **frequency of frequency** c, that is count for things that occur with frequency c.

Example: Gone Fishing (due to Joshua Goodman)

- You are fishing and you have have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel.
- How likely is it that the next species is trout?
  - $\frac{1}{18}$
- How likely is it that the next species is new (i.e. catfish or bass)?
  - According to the Good/Turing intuition: $\frac{3}{18}$

Example: Gone Fishing (continued)

- Assuming that we reserve $\frac{3}{18}$ probability mass for encountering new species, what is the probability that the next species is trout?
  - Must be less than $\frac{1}{18}$ because the "remaining probability mass" has changed after accommodating thus far unseen events.
  - How can we estimate this new scenario?

Good-Turing Estimation

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

MLE p $= \frac{0}{18}$

P^*_{GT} (unseen) $=$ N1/N $= 3/18$

MLE p $= \frac{1}{18}$

C^* (trout) $= 2 \times$ N2/N1

$= 2 \times 1/3$

P^*_{GT} (trout) $= (2/\ 3)/18 = 1/27$

How good is Good-Turing Estimation in Practice?

Bigram experiment reported by Church and Gale (1992):

- Compute the bigram counts for counts 0,...,9 in a corpus of 22 million words of AP newswire text.
- Check the counts of these bigrams in a held-out corpus of the same size.

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Figure 3.9 For all bigrams in 22 million words of AP newswire of count 0, 1, 2,. counts of these bigrams in a held-out corpus also of 22 million words.

Main Results

- Notice that the vast majority of possible bigrams has not been seen (i.e. has count zero).
- The bigrams that appear once in the training data appear on average 0.448 times in the held-out data.
- Starting with bigrams that occurred more than once, there is a rather constant relationship between the counts in the original corpus and the average counts in the held-out corpus.
- And even more importantly: this relationship is closely matched by the Good-Turing re-estimation (see the results on the next slide!)

Dan Jurafsky

Resulting Good-Turing numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	4.19
7	6.21
8	7.24
9	8.25

Kneser-Ney Smoothing

Kneser-Ney has its roots in a method called *absolute discounting*. Recall that discounting of the counts for frequent n-grams is necessary to save some probability mass for the smoothing algorithm to distribute to the unseen n-grams.

$$P_{AbsoluteDiscounting}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i) \tag{3.30}$$

Kneser-Ney Smoothing

The Kneser-Ney intuition is to base our estimate of $P_{CONTINUATION}$ on the number of different contexts word w has appeared in, that is, the number of bigram types it completes.

$$P_{CONTINUATION}(w) \propto \mid \{v : C(vw) > 0\} \mid \tag{3.31}$$

To turn this count into a probability, we normalize by the total number of word bigram types. In summary:

$$P_{CONTINUATION}(w) = \frac{\mid \{v : C(vw) > 0\} \mid}{\mid \{(u', w') : C(u'w') > 0\} \mid} \tag{3.32}$$

Interpolated Kneser-Ney Smoothing

The final equation for Interpolated Kneser-Ney smoothing for bigrams is then:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{CONTINUATION}(w_i) \tag{3.35}$$

Normalizing Constant λ

λ is a normalizing constant that is used to distribute the discounted probability mass.

$$\lambda(w_{i-1}) = \frac{d}{\sum_{\nu} C(w_{i-1}\nu)} \mid \{w : C(w_{i-1}w) > 0\} \mid \qquad (3.36)$$

Interpolated Kneser-Ney Smoothing

The general recursive formulation is as follows:

$$P_{KN}(w_i \mid w_{i-n+1:i-1}) = \frac{max(c_{KN}(w_{i-n+1:i}) - d, 0)}{\sum_{\nu} c_{KN}(w_{i-n+1:i-1}\nu)} + \lambda(w_{i-n+1:i-1}) P_{KN}(w_i \mid w_{i-n+2:i-1}) \tag{3.37}$$

At the termination of the recursion, unigrams are interpolated with the uniform distribution, where the parameter ϵ is the empty string:

$$P_{KN}(w) = \frac{max(c_{KN}(w - d, 0)}{\sum_{w'} c_{KN}(w')} + \lambda(\epsilon)\frac{1}{V} \tag{3.39}$$

Huge Language Models

- Using webdata, extremely large language models can be built.
- Examples:
  - the Web 1 Trillion 5-gram corpus released by Google
  - the 800 billion Google Books Ngram corpora
  - the balanced 1-billion-words COCA corpus (see slide above)

Example 4-grams from Google Web Corpus

4-gram	Count
serve as the incoming	92
serve as the incubator	99
serve as the independent	794
serve as the index	223
serve as the indication	72
serve as the indicator	120
serve as the indicators	45

Efficiency Matters for Huge Language Models

- Efficient storage and retrieval are important
- Examples of efficiency measures taken
  - Representing words by a 64-bit hash number
  - n-grams stored as reverse tries
  - Building approximative models
  - Using Bloom filters
  - Pruning low frequency n-grams
  - Using Stupid Backoff

Stupid Backoff

Stupid backoff gives up the idea of trying to make the language model a true probability distribution. There is no discounting of the higher-order probabilities. This algorithm does not produce a probability distribution. Brants et al. (2007) refer to it as S (where S can be either mnemonic for score or for "stupid backoff").

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{count(w_{i-k+1}^{i})}{count(w_{i-k+1}^{i-1})} & \text{if } count(w_{i-k+1}^{i}) > 0 \\ \lambda S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases} \quad (3.40)$$