

N-gram language models

Statistical Natural Language Processing N-gram Language Models

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2020

- A **language model** answers the question *how likely is a sequence of words in a given language?*
- They assign scores, typically probabilities, to sequences (of words, letters, ...)
- **n-gram** language models are the ‘classical’ approach to language modeling
- The main idea is to estimate probabilities of sequences, using the probabilities of words given a limited history
- As a bonus we get the answer for *what is the most likely word given previous words?*

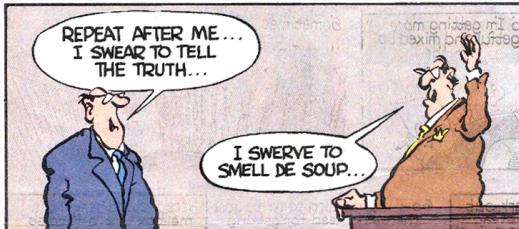
N-grams in practice: spelling correction

- How would a spell checker know that there is a spelling error in the following sentence?
I like pizza wit spinach
- Or this one?
Zoo animals on the lose

We want:

$$\begin{aligned} P(\text{I like pizza with spinach}) &> P(\text{I like pizza wit spinach}) \\ P(\text{Zoo animals on the loose}) &> P(\text{Zoo animals on the lose}) \end{aligned}$$

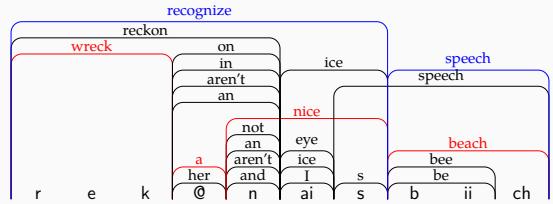
Speech recognition gone wrong



Speech recognition gone wrong



N-grams in practice: speech recognition



We want:

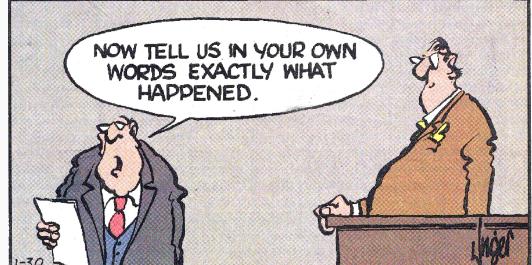
$$P(\text{recognize speech}) > P(\text{wreck a nice beach})$$

* Reproduced from Shillcock (1995)

Speech recognition gone wrong

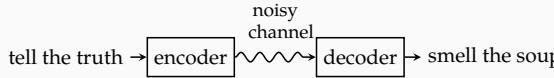


Speech recognition gone wrong



What went wrong?

Recap: noisy channel model



- We want $P(u | A)$, probability of the utterance given the acoustic signal
- The model of the noisy channel gives us $P(A | u)$
- We can use Bayes' formula

$$P(u | A) = \frac{P(A | u)P(u)}{P(A)}$$

- $P(u)$, probabilities of utterances, come from a language model

Our aim

We want to solve two related problems:

- Given a sequence of words $w = (w_1 w_2 \dots w_m)$, what is the probability of the sequence $P(w)$?
(machine translation, automatic speech recognition, spelling correction)
- Given a sequence of words $w_1 w_2 \dots w_{m-1}$, what is the probability of the next word $P(w_m | w_1 \dots w_{m-1})$?
(predictive text)



Assigning probabilities to sentences applying the chain rule

- The solution is to *decompose*

We use probabilities of parts of the sentence (words) to calculate the probability of the whole sentence

- Using the chain rule of probability (without loss of generality), we can write

$$\begin{aligned} P(w_1, w_2, \dots, w_m) &= P(w_2 | w_1) \\ &\quad \times P(w_3 | w_1, w_2) \\ &\quad \times \dots \\ &\quad \times P(w_m | w_1, w_2, \dots, w_{m-1}) \end{aligned}$$

Example: bigram probabilities of a sentence with first-order Markov assumption

$$\begin{aligned} P(\text{I like pizza with spinach}) &= P(\text{like} | \text{I}) \\ &\quad \times P(\text{pizza} | \text{like}) \\ &\quad \times P(\text{with} | \text{pizza}) \\ &\quad \times P(\text{spinach} | \text{with}) \end{aligned}$$

- Now, hopefully, we can count them in a corpus

More applications for language models

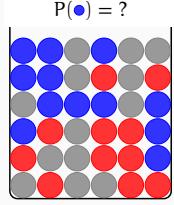
- Spelling correction
- Speech recognition
- Machine translation
- Predictive text
- Text recognition (OCR, handwritten)
- Information retrieval
- Question answering
- Text classification
- ...

Assigning probabilities to sentences

count and divide?

How do we calculate the probability a sentence like
 $P(\text{I like pizza with spinach})$

- Can we count the occurrences of the sentence, and divide it by the total number of sentences (in a large corpus)?
- Short answer: No.
 - Many sentences are not observed even in very large corpora
 - For the ones observed in a corpus, probabilities will not reflect our intuitions, or will not be useful in most applications



Example: applying the chain rule

$$\begin{aligned} P(\text{I like pizza with spinach}) &= P(\text{like} | \text{I}) \\ &\quad \times P(\text{pizza} | \text{like}) \\ &\quad \times P(\text{with} | \text{like pizza}) \\ &\quad \times P(\text{spinach} | \text{like pizza with}) \end{aligned}$$

- Did we solve the problem?
- Not really, the last term is equally difficult to estimate

Maximum-likelihood estimation (MLE)

- The MLE of n-gram probabilities is based on their frequencies in a corpus
- We are interested in conditional probabilities of the form: $P(w_i | w_1, \dots, w_{i-1})$, which we estimate using

$$P(w_i | w_{i-n+1}, \dots, w_{i-1}) = \frac{C(w_{i-n+1} \dots w_i)}{C(w_{i-n+1} \dots w_{i-1})}$$

where, $C()$ is the frequency (count) of the sequence in the corpus.

- For example, the probability $P(\text{like} | \text{I})$ would be

$$\begin{aligned} P(\text{like} | \text{I}) &= \frac{C(\text{I like})}{C(\text{I})} \\ &= \frac{\text{number of times I like occurs in the corpus}}{\text{number of times I occurs in the corpus}} \end{aligned}$$

MLE estimation of an n-gram language model

An n-gram model conditioned on $n - 1$ previous words.

unigram	$P(w_i) = \frac{C(w_i)}{N}$
bigram	$P(w_i) = P(w_i w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
trigram	$P(w_i) = P(w_i w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}$

Parameters of an n-gram model
are these conditional probabilities.

Unigram probability of a sentence

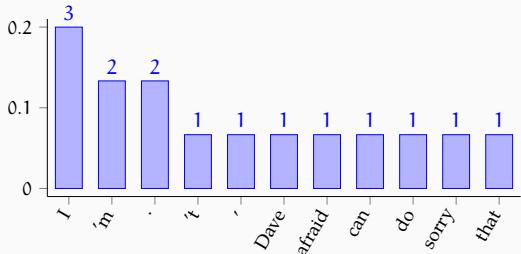
Unigram counts

I	3	,	1	afraid	1	do	1
'm	2	Dave	1	can	1	that	1
sorry	1	.	2	't	1		

$$\begin{aligned} P(I \text{ 'm sorry , Dave .}) &= P(I) \times P('m) \times P(sorry) \times P(.) \times P(Dave) \times P(.) \\ &= \frac{3}{15} \times \frac{2}{15} \times \frac{1}{15} \times \frac{1}{15} \times \frac{1}{15} \times \frac{2}{15} \\ &= 0.00000105 \end{aligned}$$

- $P(, 'm I . sorry Dave) = ?$
- Where did all the probability mass go?
- What is the most likely sentence according to this model?

Unigram probabilities



Zipf's law – a short divergence

The frequency of a word is inversely proportional to its rank:

$$\text{rank} \times \text{frequency} = k \quad \text{or} \quad \text{frequency} \propto \frac{1}{\text{rank}}$$

- This is a reoccurring theme in (computational) linguistics: most linguistic units follow more-or-less a similar distribution
- Important consequence for us (in this lecture):
 - even very large corpora will *not* contain some of the words (or n-grams)
 - there will be many low-probability events (words/n-grams)



Unigrams

Unigrams are simply the single words (or tokens).

A small corpus

I'm sorry , Dave .
I'm afraid I can't do that .

When tokenized, we have 15 tokens, and 11 types.

Unigram counts

I	3	,	1	afraid	1	do	1
'm	2	Dave	1	can	1	that	1
sorry	1	.	2	't	1		

Traditionally, can't is tokenized as can,n't (similar to have,n't, is,n't etc.), but for our purposes can,t is more readable.

N-gram models define probability distributions

- An n-gram model defines a probability distribution over words

word	prob
I	0.200
'm	0.133
.	0.133
't	0.067
,	0.067
Dave	0.067
afraid	0.067
can	0.067
do	0.067
sorry	0.067
that	0.067
	1.000

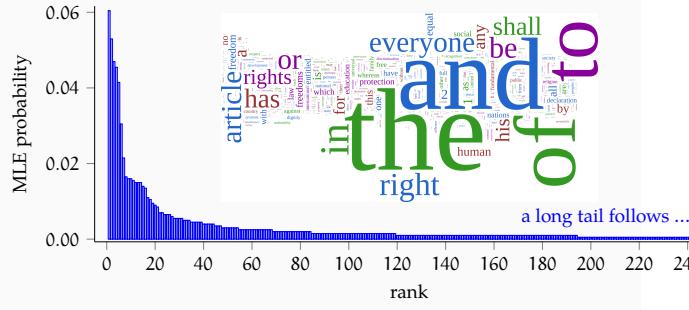
- They also define probability distributions over word sequences of equal size. For example (length 2),

$$\sum_{w \in V} \sum_{v \in V} P(w)P(v) = 1$$

- What about sentences?

Unigram probabilities in a (slightly) larger corpus

MLE probabilities in the Universal Declaration of Human Rights



Bigrams

Bigrams are overlapping sequences of two tokens.

I 'm sorry , Dave .
I 'm afraid I can 't do that .

Bigram counts

ngram	freq	ngram	freq	ngram	freq	ngram	freq
I'm	2	,Dave	1	afraid I	1	n't do	1
'm sorry	1	Dave .	1	I can	1	do that	1
sorry ,	1	'm afraid	1	can 't	1	that .	1

- What about the bigram ' . I '?

Sentence boundary markers

If we want sentence probabilities, we need to mark them.

$\langle s \rangle$ I 'm sorry , Dave . $\langle /s \rangle$
 $\langle s \rangle$ I 'm afraid I can 't do that . $\langle /s \rangle$

- The bigram ' $\langle s \rangle$ I' is not the same as the unigram 'I'
Including $\langle s \rangle$ allows us to predict likely words at the beginning of a sentence
- Including $\langle /s \rangle$ allows us to assign a proper probability distribution to sentences

Bigram probabilities

$w_1 w_2$	$C(w_1 w_2)$	$C(w_1)$	$P(w_1 w_2)$	$P(w_1)$	$P(w_2 w_1)$	$P(w_2)$
$\langle s \rangle$ I	2	2	0.12	0.12	1.00	0.18
I 'm	2	3	0.12	0.18	0.67	0.12
'm sorry	1	2	0.06	0.12	0.50	0.06
sorry ,	1	1	0.06	0.06	1.00	0.06
, Dave	1	1	0.06	0.06	1.00	0.06
Dave .	1	1	0.06	0.06	1.00	0.12
'm afraid	1	2	0.06	0.12	0.50	0.06
afraid I	1	1	0.06	0.06	1.00	0.18
I can	1	3	0.06	0.18	0.33	0.06
can 't	1	1	0.06	0.06	1.00	0.06
n't do	1	1	0.06	0.06	1.00	0.06
do that	1	1	0.06	0.06	1.00	0.06
that .	1	1	0.06	0.06	1.00	0.12
. $\langle /s \rangle$	2	2	0.12	0.12	1.00	0.12

Unigram vs. bigram probabilities

in sentences and non-sentences

w	I	'm	sorry	,	Dave	.	
P_{uni}	0.18	0.12	0.06	0.06	0.06	0.12	4.97×10^{-7}
P_{bi}	1.00	0.67	0.50	1.00	1.00	1.00	0.33

w	,	'm	I	.	sorry	Dave	
P_{uni}	0.06	0.12	0.18	0.12	0.06	0.06	4.97×10^{-7}
P_{bi}	0.00	0.00	0.00	0.00	0.00	0.00	0.00

w	I	'm	afraid	,	Dave	.	
P_{uni}	0.18	0.12	0.06	0.06	0.06	0.12	4.97×10^{-7}
P_{bi}	1.00	0.67	0.50	0.00	1.00	1.00	0.00

Trigrams

$\langle s \rangle$ $\langle s \rangle$ I 'm sorry , Dave . $\langle /s \rangle$
 $\langle s \rangle$ $\langle s \rangle$ I 'm afraid I can 't do that . $\langle /s \rangle$

Trigram counts					
ngram	freq	ngram	freq	ngram	freq
$\langle s \rangle$ $\langle s \rangle$ I	2	do that .	1	that . $\langle /s \rangle$	1
$\langle s \rangle$ I 'm	2	I 'm sorry	1	'm sorry ,	1
sorry , Dave	1	, Dave .	1	Dave . $\langle /s \rangle$	1
I 'm afraid	1	'm afraid I	1	afraid I can	1
I can 't	1	can 't do	1	't do that	1

- How many n-grams are there in a sentence of length m?

Calculating bigram probabilities

recap with some more detail

We want to calculate $P(w_2 | w_1)$. From the chain rule:

$$P(w_2 | w_1) = \frac{P(w_1, w_2)}{P(w_1)}$$

and, the MLE

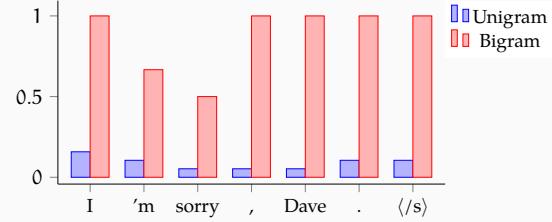
$$P(w_2 | w_1) = \frac{\frac{C(w_1 w_2)}{N}}{\frac{C(w_1)}{N}} = \frac{C(w_1 w_2)}{C(w_1)}$$

$P(w_2 | w_1)$ is the probability of w_2 given the previous word is w_1

$P(w_1, w_2)$ is the probability of the sequence $w_1 w_2$

$P(w_1)$ is the probability of w_1 occurring as the first item in a bigram,
not its unigram probability

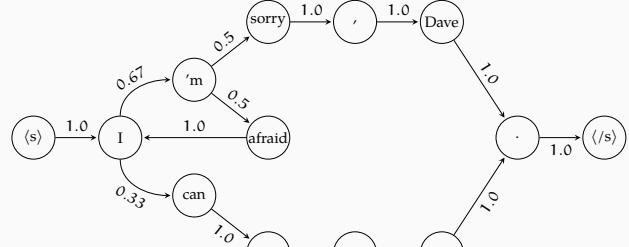
Sentence probability: bigram vs. unigram



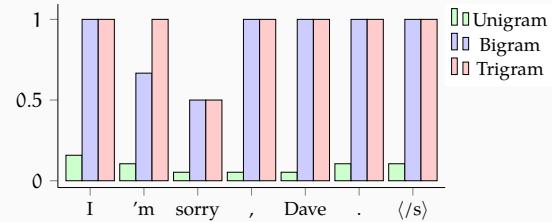
$$P_{uni}(\langle s \rangle \text{ I 'm sorry , Dave . } \langle /s \rangle) = 2.83 \times 10^{-9}$$

$$P_{bi}(\langle s \rangle \text{ I 'm sorry , Dave . } \langle /s \rangle) = 0.33$$

Bigram models as weighted finite-state automata



Trigram probabilities of a sentence



$$P_{uni}(I \text{ 'm sorry , Dave . } \langle /s \rangle) = 2.83 \times 10^{-9}$$

$$P_{bi}(I \text{ 'm sorry , Dave . } \langle /s \rangle) = 0.33$$

$$P_{tri}(I \text{ 'm sorry , Dave . } \langle /s \rangle) = 0.50$$

Short detour: colorless green ideas

But it must be recognized that the notion 'probability of a sentence' is an entirely useless one, under any known interpretation of this term. — Chomsky (1968)

- The following 'sentences' are categorically different:
 - Furiously sleep ideas green colorless
 - Colorless green ideas sleep furiously
- Can n-gram models model the difference?
- Should n-gram models model the difference?

What do n-gram models model?

- Some morphosyntax: the bigram 'ideas are' is (much more) likely than 'ideas is'
- Some semantics: 'bright ideas' is more likely than 'green ideas'
- Some cultural aspects of everyday language: 'Chinese food' is more likely than 'British food'
- more aspects of 'usage' of language

How to test n-gram models?

Extrinsic: improvement of the target application due to the language model:

- Speech recognition accuracy
- BLEU score for machine translation
- Keystroke savings in predictive text applications

Intrinsic: the higher the probability assigned to a test set better the model. A few measures:

- Likelihood
- (cross) entropy
- perplexity

Like any ML method, test set has to be different than training set.

Intrinsic evaluation metrics: cross entropy

- Cross entropy of a language model on a test set w is

$$H(w) = -\frac{1}{N} \sum_{w_i} \log_2 \hat{P}(w_i)$$

- The lower the cross entropy, the better the model
- Cross entropy is not sensitive to the test-set size

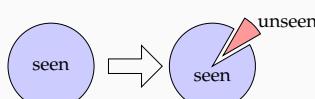
Reminder: Cross entropy is the bits required to encode the data coming from P using another (approximate) distribution \hat{P} .

$$H(P, Q) = -\sum_x P(x) \log \hat{P}(x)$$

What do we do with unseen n-grams?

...and other issues with MLE estimates

- Words (and word sequences) are distributed according to the Zipf's law: *many words are rare*.
- MLE will assign 0 probabilities to unseen words, and sequences containing unseen words
- Even with non-zero probabilities, MLE *overfits* the training data
- One solution is **smoothing**: take some probability mass from known words, and assign it to unknown words



Intrinsic evaluation metrics: likelihood

- Likelihood of a model M is the probability of the (test) set w given the model

$$\mathcal{L}(M | w) = P(w | M) = \prod_{s \in w} P(s)$$

- The higher the likelihood (for a given test set), the better the model
- Likelihood is sensitive to the test set size
- Practical note: (minus) log likelihood is used more commonly, because of ease of numerical manipulation

Intrinsic evaluation metrics: perplexity

- Perplexity is a more common measure for evaluating language models

$$PP(w) = 2^{H(w)} = P(w)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w)}}$$

- Perplexity is the average branching factor
- Similar to cross entropy
 - lower better
 - not sensitive to test set size

Laplace smoothing (Add-one smoothing)

- The idea (from 1790): add one to all counts
- The probability of a word is estimated by

$$P_{+1}(w) = \frac{C(w)+1}{N+V}$$

N number of word tokens
 V number of word types - the size of the vocabulary

- Then, probability of an unknown word is:

$$\frac{0+1}{N+V}$$

Laplace smoothing

for n-grams

- The probability of a bigram becomes

$$P_{+1}(w_i w_{i-1}) = \frac{C(w_i w_{i-1}) + 1}{N + V^2}$$

- and, the conditional probability

$$P_{+1}(w_i | w_{i-1}) = \frac{C(w_{i-1} w_i) + 1}{C(w_{i-1}) + V}$$

- In general

$$P_{+1}(w_{i-n+1}^i) = \frac{C(w_{i-n+1}^i) + 1}{N + V^n}$$

$$P_{+1}(w_{i-n+1}^i | w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^i) + 1}{C(w_{i-n+1}^{i-1}) + V}$$

MLE vs. Laplace probabilities

probabilities of sentences and non-sentences (based on the bigram model)

w	I	'm	sorry	,	Dave	.	⟨/s⟩	
P _{MLE}	1.00	0.67	0.50	1.00	1.00	1.00	1.00	0.33
P ₊₁	0.19	0.18	0.13	0.13	0.13	0.13	0.19	1.84×10^{-6}

w	,	'm	I	.	sorry	Dave	⟨/s⟩	
P _{MLE}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P ₊₁	0.03	0.03	0.03	0.03	0.03	0.03	0.03	1.17×10^{-12}

w	I	'm	afraid	,	Dave	.	⟨/s⟩	
P _{MLE}	1.00	0.67	0.50	0.00	1.00	1.00	1.00	0.00
P ₊₁	0.19	0.18	0.13	0.03	0.13	0.13	0.19	4.45×10^{-7}

Lidstone correction

(Add- α smoothing)

- A simple improvement over Laplace smoothing is adding α instead of 1

$$P_{+\alpha}(w_{i-n+1}^i | w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^i) + \alpha}{C(w_{i-n+1}^{i-1}) + \alpha V}$$

- With smaller α values, the model behaves similar to MLE, it overfits: it has high variance
- Larger α values reduce overfitting/variance, but result in large bias

We need to tune α like any other hyperparameter.

Good-Turing smoothing

- Estimate the probability mass to be reserved for the novel n-grams using the observed n-grams
- Novel events in our training set is the ones that occur once

$$p_0 = \frac{n_1}{n}$$

where n_1 is the number of distinct n-grams with frequency 1 in the training data

- Now we need to discount this mass from the higher counts
- The probability of an n-gram that occurred r times in the corpus is

$$(r + 1) \frac{n_{r+1}}{n_r n}$$

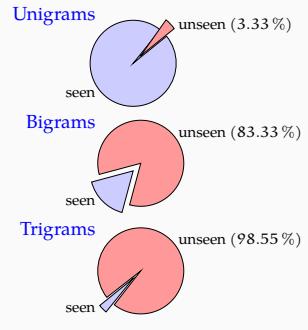
Bigram probabilities

MLE vs. Laplace smoothing

w ₁ w ₂	C ₊₁	P _{MLE} (w ₁ w ₂)	P ₊₁ (w ₁ w ₂)	P _{MLE} (w ₂ w ₁)	P ₊₁ (w ₂ w ₁)
⟨s⟩ I	3	0.118	0.019	1.000	0.188
I'm	3	0.118	0.019	0.667	0.176
'm sorry	2	0.059	0.012	0.500	0.125
sorry ,	2	0.059	0.012	1.000	0.133
, Dave	2	0.059	0.012	1.000	0.133
Dave .	2	0.059	0.012	1.000	0.133
'm afraid	2	0.059	0.012	0.500	0.125
afraid I	2	0.059	0.012	1.000	0.133
I can	2	0.059	0.012	0.333	0.118
can 't	2	0.059	0.012	1.000	0.133
n't do	2	0.059	0.012	1.000	0.133
do that	2	0.059	0.012	1.000	0.133
that .	2	0.059	0.012	1.000	0.133
. ⟨/s⟩	3	0.118	0.019	1.000	0.188
\sum			1.000	0.193	

How much probability mass does +1 smoothing steal?

- Laplace smoothing reserves probability mass proportional to the size of the vocabulary
- This is just too much for large vocabularies and higher order n-grams
- Note that only very few of the higher level n-grams (e.g., trigrams) are possible



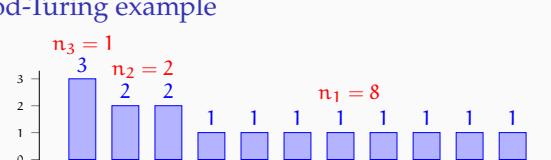
Absolute discounting



- An alternative to the additive smoothing is to reserve an explicit amount of probability mass, ϵ , for the unseen events
- The probabilities of known events has to be re-normalized
- How do we decide what ϵ value to use?

Good-Turing example

Good-Turing example



$$P_{GT}(\text{the}) + P_{GT}(\text{a}) + \dots = \frac{8}{15}$$

$$P_{GT}(\text{that}) = P_{GT}(\text{do}) = \dots = \frac{2 \times 2}{15 \times 8}$$

$$P_{GT}('m) = P_{GT}(.) = \frac{3 \times 1}{15 \times 2}$$

Issues with Good-Turing discounting

With some solutions

- Zero counts: we cannot assign probabilities if $n_{r+1} = 0$
- The estimates of some of the frequencies of frequencies are unreliable
- A solution is to replace n_r with smoothed counts z_r
- A well-known technique (simple Good-Turing) for smoothing n_r is to use linear interpolation

$$\log z_r = a + b \log r$$

Back-off and interpolation

The general idea is to fall-back to lower order n-gram when estimation is unreliable

- Even if,

$$C(\text{black squirrel}) = C(\text{black wug}) = 0$$

it is unlikely that

$$C(\text{squirrel}) = C(\text{wug})$$

in a reasonably sized corpus

Interpolation

Interpolation uses a linear combination:

$$P_{\text{int}}(w_i | w_{i-1}) = \lambda P(w_i | w_{i-1}) + (1 - \lambda)P(w_i)$$

In general (recursive definition),

$$P_{\text{int}}(w_i | w_{i-n+1}^{i-1}) = \lambda P(w_i | w_{i-n+1}^{i-1}) + (1 - \lambda)P_{\text{int}}(w_i | w_{i-n+2}^{i-1})$$

- $\sum \lambda_i = 1$
- Recursion terminates with
 - either smoothed unigram counts
 - or uniform distribution $\frac{1}{V}$

Katz back-off

A popular back-off method is Katz back-off:

$$P_{\text{Katz}}(w_i | w_{i-n+1}^{i-1}) = \begin{cases} P^*(w_i | w_{i-n+1}^{i-1}) & \text{if } C(w_{i-n+1}^{i-1}) > 0 \\ \alpha_{w_{i-n+1}^{i-1}} P_{\text{Katz}}(w_i | w_{i-n+2}^{i-1}) & \text{otherwise} \end{cases}$$

- $P^*(\cdot)$ is the Good-Turing discounted probability estimate (only for n-grams with small counts)
- $\alpha_{w_{i-n+1}^{i-1}}$ makes sure that the back-off probabilities sum to the discounted amount
- α is high for frequent contexts. So, hopefully,

$$\alpha_{\text{black}} P(\text{squirrel}) > \alpha_{\text{wuggy}} P(\text{squirrel})$$

$$P(\text{squirrel} | \text{black}) > P(\text{squirrel} | \text{wuggy})$$



Not all (unknown) n-grams are equal

- Let's assume that `black squirrel` is an unknown bigram
- How do we calculate the smoothed probability

$$P_{+1}(\text{squirrel} | \text{black}) = \frac{0 + 1}{C(\text{black}) + V}$$

- How about `black wug`?

$$P_{+1}(\text{black wug}) = P_{+1}(\text{wug} | \text{black}) = \frac{0 + 1}{C(\text{black}) + V}$$

- Would it make a difference if we used a better smoothing method (e.g., Good-Turing?)

Back-off

Back-off uses the estimate if it is available, ‘backs off’ to the lower order n-gram(s) otherwise:

$$P(w_i | w_{i-1}) = \begin{cases} P^*(w_i | w_{i-1}) & \text{if } C(w_{i-1} w_i) > 0 \\ \alpha P(w_i) & \text{otherwise} \end{cases}$$

where,

- $P^*(\cdot)$ is the discounted probability
- α makes sure that $\sum P(w)$ is the discounted amount
- $P(w_i)$, typically, smoothed unigram probability

Not all contexts are equal

- Back to our example: given both bigrams

- `black squirrel`
- `wuggy squirrel`

are unknown, the above formulations assign the same probability to both bigrams

- To solve this, the back-off or interpolation parameters (α or λ) are often conditioned on the context
- For example,

$$P_{\text{int}}(w_i | w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P(w_i | w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{\text{int}}(w_i | w_{i-n+2}^{i-1})$$

Kneser-Ney interpolation: intuition

- Use absolute discounting for the higher order n-gram
- Estimate the lower order n-gram probabilities based on the probability of the target word occurring in a new context
- Example:
I can't see without my reading ____.
- It turns out the word `Francisco` is more frequent than `glasses` (in the typical English corpus, PTB)
- But `Francisco` occurs only in the context `San Francisco`
- Assigning probabilities to unigrams based on the number of unique contexts they appear makes `glasses` more likely

Kneser-Ney interpolation

for bigrams

$$P_{KN}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - D}{C(w_i)} + \lambda_{w_{i-1}} \frac{\sum_w |v| C(vw_i) > 0]}{\sum_w |v| C(vw) > 0}}$$

(Absolute discount)

Unique contexts w_i appears
All unique contexts

- λ s make sure that the probabilities sum to 1
- The same idea can be applied to back-off as well (interpolation seems to work better)

Cluster-based n-grams

- The idea is to cluster the words, and fall-back (back-off or interpolate) to the cluster
- For example,
 - a clustering algorithm is likely to form a cluster containing words for food, e.g., {apple, pear, broccoli, spinach}
 - if you have never seen eat your broccoli, estimate
$$P(\text{broccoli}|\text{eat your}) = P(\text{FOOD}|\text{eat your}) \times P(\text{broccoli}|\text{FOOD})$$
- Clustering can be
 - hard a word belongs to only one cluster (simplifies the model)
 - soft words can be assigned to clusters probabilistically (more flexible)

Modeling sentence types

- Another way to improve a language model is to condition on the sentence types
- The idea is different types of sentences (e.g., ones related to different topics) have different behavior
- Sentence types are typically based on clustering
- We create multiple language models, one for each sentence type
- Often a ‘general’ language model is used, as a fall-back

Structured language models

- Another possibility is using a generative parser
- Parses try to explicitly model (good) sentences
- Parses naturally capture long-distance dependencies
- Parses require much more computational resources than the n-gram models
- The improvements are often small (if any)

Some shortcomings of the n-gram language models

The n-gram language models are simple and successful, but ...

- They cannot handle long-distance dependencies:
In the last race, the horse he bought last year finally ____.
- The success often drops in morphologically complex languages
- The smoothing methods are often ‘a bag of tricks’
- They are highly sensitive to the training data: you do not want to use an n-gram model trained on business news for medical texts

Skipping

- The contexts
 - boring | the lecture was
 - boring | (the) lecture yesterday was
 are completely different for an n-gram model
- A potential solution is to consider contexts with gaps, ‘skipping’ one or more words
- We would, for example model $P(e | abcd)$ with a combination (e.g., interpolation) of
 - $P(e | abc_)$
 - $P(e | ab_d)$
 - $P(e | a_cd)$
 - ...

Caching

- If a word is used in a document, its probability of being used again is high
- Caching models condition the probability of a word, to a larger context (besides the immediate history), such as
 - the words in the document (if document boundaries are marked)
 - a fixed window around the word

Maximum entropy models

- We can fit a logistic regression ‘max-ent’ model predicting $P(w | \text{context})$
- Main advantage is to be able to condition on arbitrary features

Adjusted counts

Sometimes it is instructive to see the ‘effective count’ of an n-gram under the smoothing method.

For Good-Turing smoothing, the updated count, r^* is

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

- novel items: n_1
- singletons: $\frac{2 \times n_2}{n_1}$
- doubletons: $\frac{3 \times n_3}{n_2}$
- ...

A quick summary

Markov assumption

- Our aim is to assign probabilities to sentences
 $P(I'm\ sorry, Dave.) = ?$

Problem: We cannot just count & divide

- Most sentences are rare: no (reliable) way to count their occurrences
- Sentence-internal structure tells a lot about its probability

Solution: Divide up, simplify with a Markov assumption

$$P(I'm\ sorry, Dave) =$$

$$P(I|s)P(m|I)P(sorry|m)P(.)P(sorry)P(Dave|.)P(.)P(Dave)P(/s|.)$$

Now we can count the parts (n-grams), and estimate their probability with MLE.

A quick summary

Smoothing

Problem The MLE assigns 0 probabilities to unobserved n-grams, and any sentence containing unobserved n-grams. In general, it *overfits*

Solution Reserve some probability mass for unobserved n-grams
 Additive smoothing add α to every count

$$P_{+\alpha}(w_{i-n+1}^i | w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^i) + \alpha}{C(w_{i-n+1}^{i-1}) + \alpha V}$$

- Discounting
- reserve a fixed amount of probability mass to unobserved n-grams
 - normalize the probabilities of observed n-grams
 - (e.g., Good-Turing smoothing)



A quick summary

Back-off & interpolation

Problem if unseen we assign the same probability for

- black squirrel
- black wug

Solution Fall back to lower-order n-grams when you cannot estimate the higher-order n-gram

Back-off

$$P(w_i | w_{i-1}) = \begin{cases} P^*(w_i | w_{i-1}) & \text{if } C(w_{i-1} w_i) > 0 \\ \alpha P(w_i) & \text{otherwise} \end{cases}$$

Interpolation

$$P_{\text{int}}(w_i | w_{i-1}) = \lambda P(w_i | w_{i-1}) + (1 - \lambda) P(w_i)$$

Now $P(\text{squirrel})$ contributes to $P(\text{squirrel} | \text{black})$, it should be higher than $P(\text{wug} | \text{black})$.

A quick summary

Problems with simple back-off / interpolation

Problem if unseen, we assign the same probability for

- black squirrel
- wuggy squirrel

Solution make normalizing constants (α, λ) context dependent, higher for context n-grams that are more frequent

Back-off

$$P(w_i | w_{i-1}) = \begin{cases} P^*(w_i | w_{i-1}) & \text{if } C(w_{i-1} w_i) > 0 \\ \alpha_{i-1} P(w_i) & \text{otherwise} \end{cases}$$

Interpolation

$$P_{\text{int}}(w_i | w_{i-1}) = P^*(w_i | w_{i-1}) + \lambda_{w_{i-1}} P(w_i)$$

Now $P(\text{black})$ contributes to $P(\text{squirrel} | \text{black})$, it should be higher than $P(\text{wuggy} | \text{squirrel})$.

A quick summary

More problems with back-off / interpolation

Problem if unseen, we assign higher probability to

- reading Francisco
- than
- reading glasses

Solution Assigning probabilities to unigrams based on the number of unique contexts they appear

Francisco occurs only in *San Francisco*, but *glasses* occur in more contexts.