Language Models

(Natural Language Processing)

HND Thilini
hnd@ucsc.cmb.ac.lk

some slides from Dan Jurafsky

Example sentences

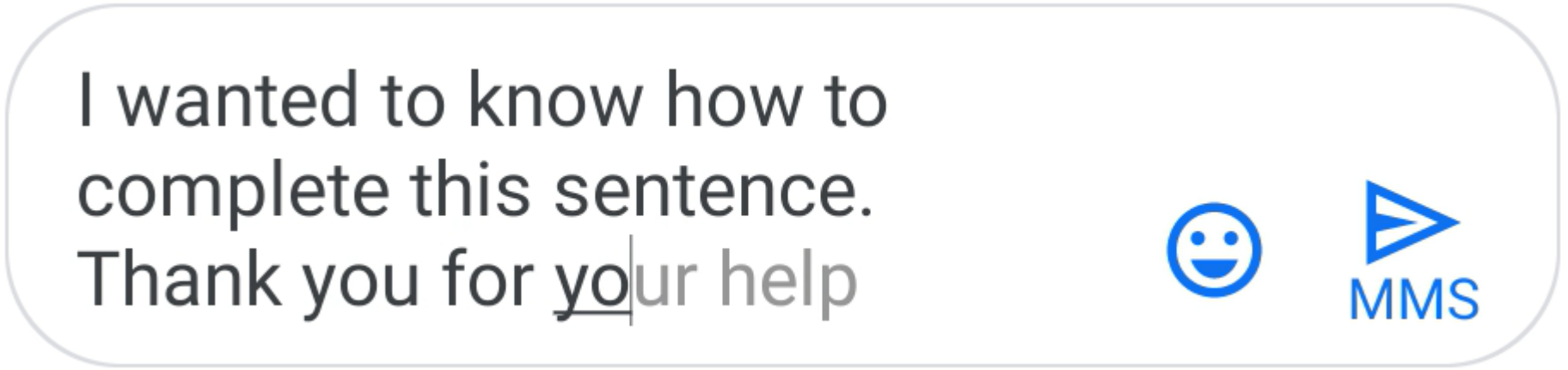
- Your help thanks
- Thank for help your you
- For your help thank you
- Thank you for your help
- Your thank help for you

Example sentences

- Your help thanks
- Thank for help your you
- For your help thank you
- Thank you for your help
- Your thank help for you

- The house is small
- Small the is house

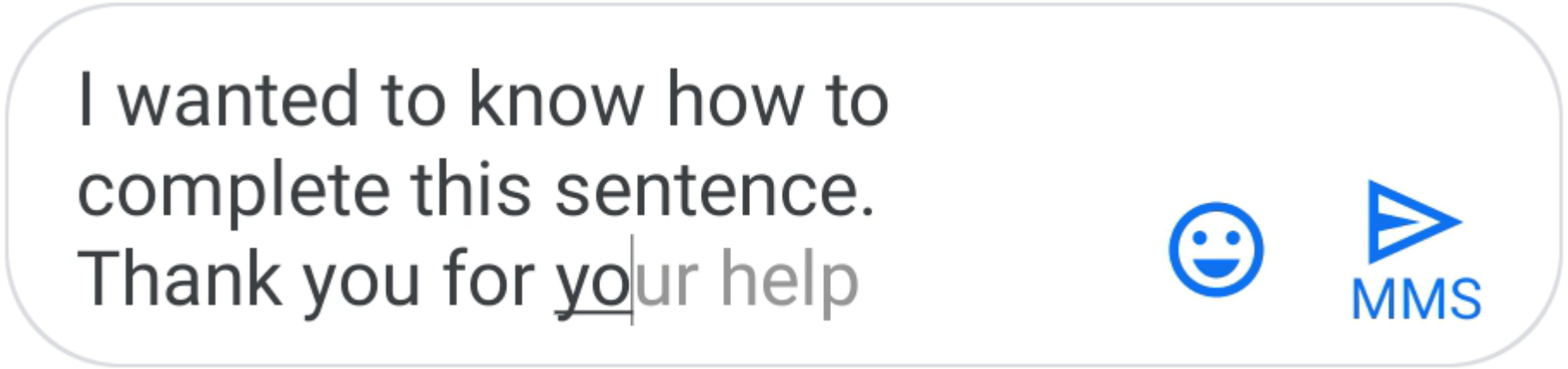
Smart Compose - Google



Haven't seen you in a while and I hope you're doing well.

Smart Compose - Google

- One of the use-cases of language models used in NLP



Haven't seen you in a while and I hope you're doing well.

What is a Language Model?

Language Models / Modeling

- It is a statistical tool that analyzes the pattern of human language for the prediction of words.
- The goal of a Language Model is to assign a probability that a sentence will occur

pLM(the house is small) > pLM(small the is house)

Language Models

- Why?
  - Machine Translation:
    - P(**high** winds tonight) > P(**large** winds tonight)
  - Spell Correction
    - The office is about fifteen **minuets** from my house
      - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - Speech Recognition
    - P(I saw a van) >> P(eyes awe of an)
  - Sentimental Analysis
  - + Summarization, question-answering, and many other NLP applications

Language Models

- **Goal:** compute the probability of a sentence or sequence of words:

  P(W) = P(w1,w2,w3,w4,w5…wn)

- **Related task**: probability of an upcoming word:
  - P(w5|w1, w2, w3, w4)
  - Conditional probability that w5 occurs, given that we know that w1, w2, w3, and w4 already occurred.
- A model that computes either of these is called a ***language model***
  - We might call this a grammar because it predicts the structure of the language, but the “language model” is the standard terminology.

Chain rule applied…

- Compute the probability of a sentence by computing the joint probability of all the words conditioned by the previous words

$$P(w_1 w_2 \cdots w_n) = \prod_i P(w_i \mid w_1 w_2 \cdots w_{i-1})$$

P(“its water is so transparent”) =

P(its) x P(water|its) x P(is|its water) x P(so|its water is)

x P(transparent|its water is so)

Computing Probabilities

- Normally, we just compute the probability that something occurred by counting its occurrences and dividing by the total number

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{\mathit{Count}(\text{its water is so transparent that the})}{\mathit{Count}(\text{its water is so transparent that})}$$

- But there are way too many English sentences in any realistic corpus for this to work!
  - We'll never see enough data

Markov Assumption

- Instead we make the simplifying Markov assumption that we can predict the next word based on only one word previous:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- or perhaps two words previous:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Andrei Markov

N-gram Models

- **Unigram Model:** The simplest case is that we predict a sentence probability just base on the probabilities of the words with no preceding words

$$P(w_1 w_2 \cdots w_n) \approx \prod_i P(w_i)$$

- **Bigram Model:** Prediction based on one previous word:

$$P(w_i \mid w_1 w_2 \cdots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

Bigrams

- Examples of bigrams are any two words that occur together
  - In the text: "*two great and powerful groups of nations*", the bigrams are; "*two great*", "*great and*", "*and powerful*", etc.
- The ***frequency*** of an n-gram is the percentage of times the n-gram occurs in all the n-grams of the corpus and could be useful in corpus statistics
  - For bigram xy:
    - Count of bigram xy / Count of all bigrams in corpus
- But in **bigram language models**, we use the *bigram probability* to predict how likely it is that the second word follows the first

N-gram Models

- We can extend to trigrams, 4-grams, 5-grams
  - Each higher number will get a more accurate model, but will be harder to find examples of the longer word sequences in the corpus
- In general this is an insufficient model of language
  - because language has long-distance dependencies:

  "The computer which I had just put into the machine room on the fifth floor crashed."
    - the last word *crashed* is not very likely to follow the word *floor*, but it is likely to be the main verb of the word *computer*
- But we can often get away with N-gram models

N-Gram probabilities

- For N-Grams, we need the conditional probability:

  P(<next word> | <preceding word sequence of length n)

  e.g. P (*the* | *They picnicked by*)

- We define this as
  - the observed frequency (count) of the whole sequence divided by the observed frequency of the preceding, or initial, sequence.
  - sometimes called the Maximum Likelihood Estimation (MLE):

$$P(\text{<next word>} \mid \text{<preceding word sequence of length n>}) = \frac{\text{Count (<preceding word sequence> <next word>)}}{\text{Count (<preceding word sequence>)}}$$

Eg: Count (*They picnicked by the*) / Count (*They picnicked by*)

Bigram Probabilities

- For bigrams, the MLE estimate is:

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

  - The count of the occurrences of the sequence w_{i-1} w_i divided by the count of the first word w_{i-1}

Example of Bigram probabilities

- Example mini-corpus of three sentences, where we have sentence detection and we include the sentence tags in order to represent the beginning and end of the sentence.

  <S> *I am Sam* </S>

  <S> *Sam I am* </S>

  <S> *I do not like green eggs and ham* </S>

- Bigram probabilities:

  P (I | <S>) (*probability that I follows* <S>) =

  P (</S> | Sam) =

  P (Sam | <S>) =

  P (Sam | am) =

  P (am | I) =

Example of Bigram probabilities

- Example mini-corpus of three sentences, where we have sentence detection and we include the sentence tags in order to represent the beginning and end of the sentence.

  <S> *I am Sam* </S>

  <S> *Sam I am* </S>

  <S> *I do not like green eggs and ham* </S>

- Bigram probabilities:

  P (I | <S>) = 2/3 = 0.67

  P (</S> | Sam) = ½ = 0.5

  P (Sam | <S>) = 1/3 = 0.33

  P (Sam | am) = ½ = 0.5

  P (am | I) = 2/3 = 0.67

Raw Bigram Counts from the corpus

- Out of 9332 sentences in Berkeley Restaurant Project corpus, showing counts that the word on the left is followed by the word on the top…

	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

Raw Bigram probabilities

- Normalize by unigrams:

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

- Result:

	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

Using N-Grams for sentences

- For a bigram grammar; $\prod_{k=1}^{n} P(w_k \mid w_{k-1})$

  - P(sentence) can be approximated by multiplying all the bigram probabilities in the sequence.

Using N-Grams for sentences

- Bigram probabilities:

  P (I | <S>) = 2/3 = 0.67 (*probability that I follows* <S>)

  P (</S> | Sam) = ½ = 0.5

  P (Sam | <S>) = 1/3 = 0.33

  P (Sam | am) = ½ = 0.5

  P (I | Sam) = 1/4 = 0.25

  P (am | I) = 2/3 = 0.67

  P (</S> | am) = 1/8 = 0.125

<S> *I am Sam* </S> =

<S> *Sam I am* </S> =

Using N-Grams for sentences

- Bigram probabilities:

  P (I | <S>) = 2/3 = 0.67 (*probability that I follows* <S>)

  P (</S> | Sam) = ½ = 0.5

  P (Sam | <S>) = 1/3 = 0.33

  P (Sam | am) = ½ = 0.5

  P (I | Sam) = 1/4 = 0.25

  P (am | I) = 2/3 = 0.67

  P (</S> | am) = 1/8 = 0.125

<S> *I am Sam* </S> = 0.67 x 0.67 x 0.5 x 0.5 = 0.1122

<S> *Sam I am* </S> = 0.33 x 0.25 x 0.67 x 0.125 = 0.0069

Using N-Grams for sentences

- More example

P(I want to eat Chinese food) =

P(I|<S>) P(want|I) P(to|want) P(eat|to)

P(Chinese|eat) P(food|Chinese)

More Bigrams from the restaurant corpus

eat on	.16	eat Thai	.03
eat some	.06	eat breakfast	.03
eat lunch	.06	eat in	.02
eat dinner	.05	eat Chinese	.02
eat at	.04	eat Mexican	.02
eat a	.04	eat tomorrow	.01
eat Indian	.04	eat dessert	.007
eat today	.03	eat British	.001

Additional Bigrams

<S> I	.25	want some	.04
<S> I'd	.06	want Thai	.01
<S> Tell	.04	to eat	.26
<S> I'm	.02	to have	.14
I want	.32	to spend	.09
I would	.29	to be	.02
I don't	.08	British food	.60
I have	.04	British restaurant	.15
want to	.65	British cuisine	.01
want a	.05	British lunch	.01

Computing Sentence Probabilities

P(I want to eat British food) =

P(I|<S>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British)

= .25×.32×.65×.26×.001×.60 = .000080

vs.

P(I want to eat Chinese food) = .00015

- Probabilities seem to capture "syntactic" facts, "world knowledge"
  - eat is often followed by a NP
  - British food is not too popular
- N-gram models can be trained by counting and normalization

Evaluating Language models

- The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves.

- Such end-to-end evaluation is called ***extrinsic evaluation***.
  - For speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.

- However, running big NLP systems end-to-end is often very expensive.

Evaluating Language models

- An ***intrinsic evaluation*** metric is one that measures the quality of a model independent of any application.
- For an intrinsic evaluation of a language model we need a *test set*.
  - the probabilities of an n-gram model come from the corpus it is trained on, the **training set**.
  - quality of an n-gram model can be measured by its performance on some unseen data called **the test** set or **test corpus**.
- In practice, we often just divide our data into
  - 80% training, and 20% test.

Evaluating Language models

- Perplexity
  - In practice we don't use raw probability as our metric for evaluating language models, but a variant called **perplexity**.

- The **perplexity** (PP) of a language model on a test set is the **inverse probability of the test set, normalized by the number of words**.
  - For a test set W = $w_1 w_2 \ldots w_N$

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

Evaluating Language models

- Perplexity

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

- Using the chain rule to expand the probability of W:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Evaluating Language models

- Perplexity
  - perplexity of W with a bigram language model:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

- Because of the inverse, the higher the conditional probability of the word sequence, the lower the perplexity.
- Minimizing perplexity is equivalent to maximizing the test set probability according to the language model.

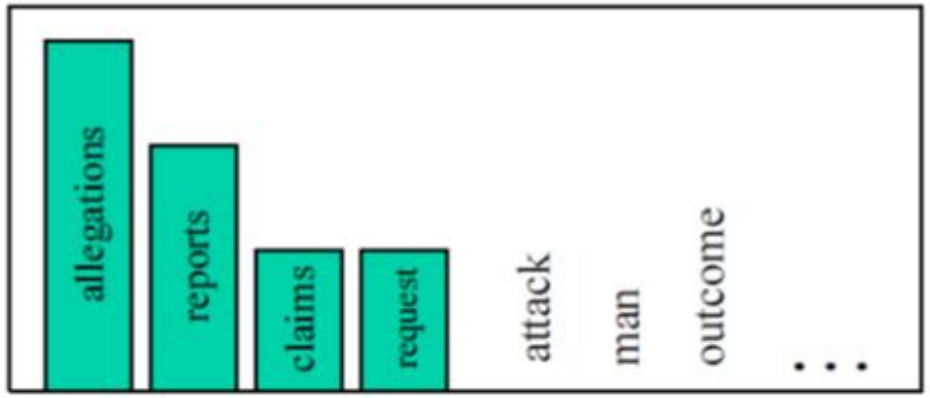
Zeros, Unseen Words & Unseen Events

- Zeros
  - things that don't ever occur in the training set but do occur in the test set
- Unseen Words
  - Every N-gram training matrix is sparse, even for very large corpora
  - There are words that don't occur in the training corpus that may occur in future text
- Unseen Events
  - words that are in our vocabulary (they are not un words) but appear in a test set in an unseen context
- Whenever a probability is 0, it will multiply the entire sequence to be 0
- Solution: estimate the likelihood of unseen N-grams and include a small probability for unseen words - ***Smoothing***

Intuition of smoothing (from Dan Klein)

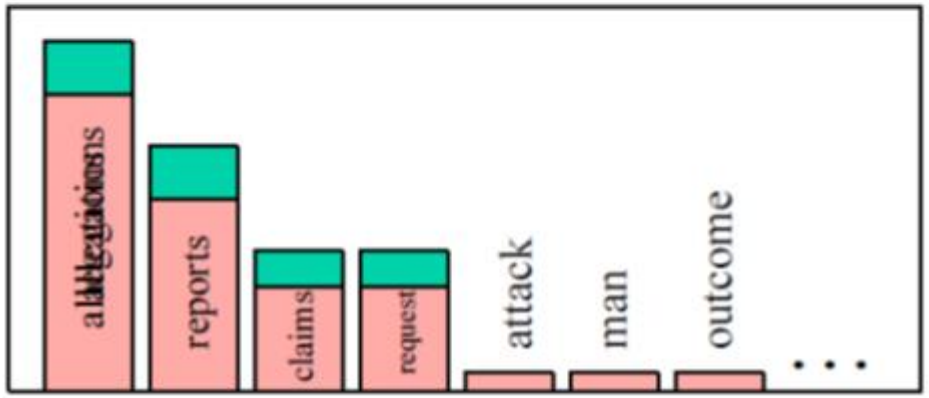
- When we have sparse statistics:

P(w | denied the)
- 3 allegations
- 2 reports
- 1 claims
- 1 request
- 7 total



- Steal probability mass to generalize better

P(w | denied the)
- 2.5 allegations
- 1.5 reports
- 0.5 claims
- 0.5 request
- 2 other
- 7 total



Smoothing

- Add-one smoothing (Laplace Smoothing)
  - Given: $P(w_n | w_{n-1}) = C(w_{n-1}w_n)/C(w_{n-1})$
  - Add 1 to each count: $P(w_n | w_{n-1}) = [C(w_{n-1}w_n) + 1] / [C(w_{n-1}) + V]$

	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 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

	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 4.5 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Add-One Smoothing

	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

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 4.6 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences.

Issue on Add-One Smoothing

- It is often convenient to reconstruct the count matrix so we can see how much a smoothing algorithm has changed the original counts.

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

	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 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

	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 4.7 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences.

Smoothing

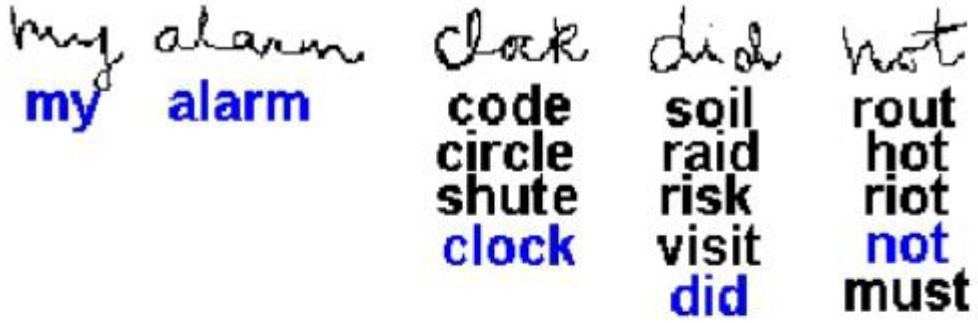
- Backoff Smoothing for higher-order N-grams
  - Notice that:
    - N-grams are more precise than (N-1) grams
    - But also, N-grams are more sparse than (N-1) grams
  - How to combine things?
    - Attempt N-grams and back-off to (N-1) if counts are not available
    - E.g. attempt prediction using 4-grams, and back-off to trigrams (or bigrams, or unigrams) if counts are not available
- More complicated techniques exist: in practice, NLP LM use Kneser-Ney smoothing

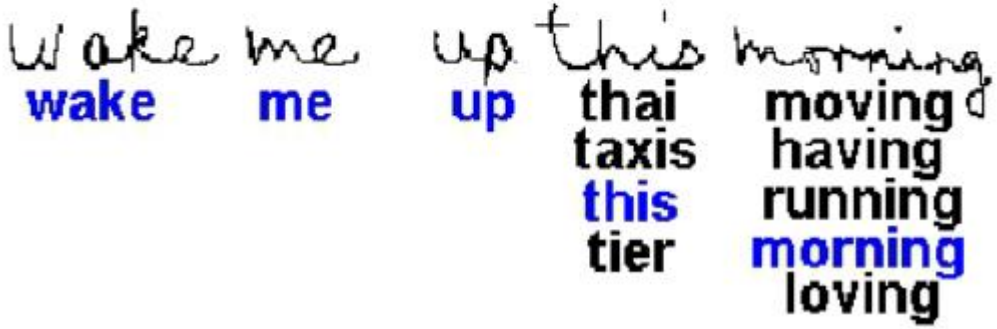
N-gram Model Applications: Spell Correction

- Frequency of spelling errors in human typed text varies
  - 0.05% of the words in carefully edited journals
  - 38% in difficult applications like telephone directory lookup
- Word-based spell correction checks each word in a dictionary/lexicon
  - Detecting spelling errors that result in non-words
  - *mesage -> message* by looking only at the **word** in isolation
- May fail to recognize an error (**real-word errors**)
  - Typographical errors e.g. *there* for *three*
  - Homonym or near-homonym e.g. *dessert* for *desert*, or *piece* for *peace*
- Use context of preceding word and language model to choose correct word
  - *Japanese Empirical Navy -> Japanese Imperial Navy*

N-gram Model Analysis of Handwritten Sentences

- Optical character recognition has higher error rates than human typists
- Lists of up to top 5 choices of the handwritten word recognizer, with correct choice highlighted
- Using language models with collocational (*alarm clock*) & syntactic (POS) information, correct sentence is extracted:





N-gram Model Applications

- Natural language processing (NLP) ...
- Chatbots and virtual assistants. ...
- Content generation and automation. ...
- Language translation. ...
- Sentiment analysis and text classification. ...
- Custom LLM model development. ...
- Question answering systems. ...
- Code generation and debugging
- etc…

Google N-gram Viewer

- **Google N-gram Viewer**
  - is an online search engine that charts the frequencies of any set of comma-delimited search strings using a yearly count of *n*-grams found in sources published between 1500 and 2019 Google's text corpora in English, Chinese (simplified), French, German, Hebrew, Italian, Russian, or Spanish.
- developed by Jon Orwant and Will Brockman and released in mid-December 2010.
- The Ngram Viewer was initially based on the 2009 edition of the Google Books Ngram Corpus.
- As of July 2020, the program supports 2009, 2012, and 2019 corpora.
- https://books.google.com/ngrams

Google N-gram Viewer

- Basic search
  - Human rights, good governance, corruption, air pollution, global warming
- Wildcard search
  - University of *
- Inflection search
  - Write_INF
- Part-of-speech Tags
  - drink_VERB, drink_NOUN
- And many more…
  - https://books.google.com/ngrams/info