CS 505: Introduction to Natural Language Processing

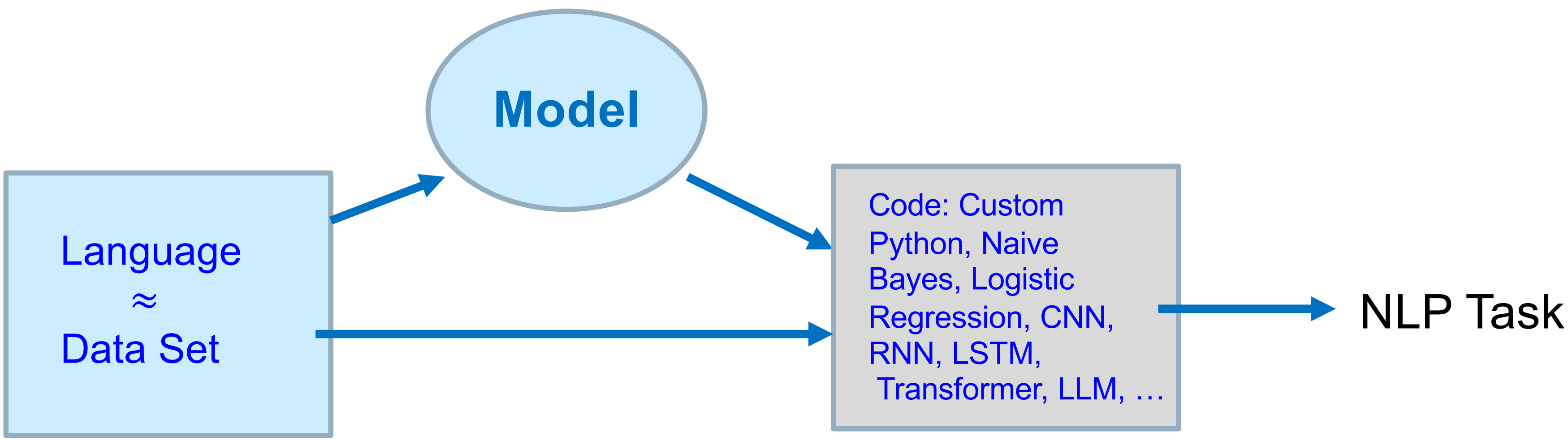
Wayne Snyder
Boston University

Lecture 4: Language Models, Bag-of-Words, N-grams, Skip-Grams

Language Models

A Language Model is a simplified representation of a language which facilitates an NLP task, where

- A language (potentially infinite) is approximated by a (finite) data set; and
- The model is a set of (simplified) assumptions about the language, embodied by the algorithms and data structures of your program.



Language Models

NLP systems rely on *models to capture knowledge* of about a language, and as a representation for texts which facilitate an NLP task.

Example: Context-Free Grammars (Chomsky, Backus-Naur)

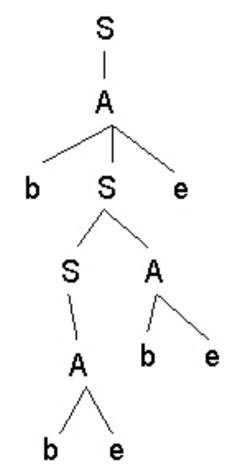
- S -> SA
  S -> A
  A -> bSe
  A -> be

Example: b and e matched as parentheses

derivation of bbebee

S
A
bSe
bSAe
bAAe
bbeAe
bbebee

hierarchical parse tree



(But we won't be using CNFs in this course!)

Language Modeling: Word Representations

Before diving into the subject of Language Models, let’s prepare a bit by talking about an essential component of language modeling…

Last time we discussed how to represent characters, as integer ASCII codes in the range [0 .. 127]. But how do we represent words?

Bad idea: word = sequence of ASCII codes

Why is this bad?

- Variable length,
- Information contains confusing correspondences:

  “to” very similar to “too” “dog” same chars as “god”

(Neural networks will find these difficult to learn.)

Language Modeling: Word Representations

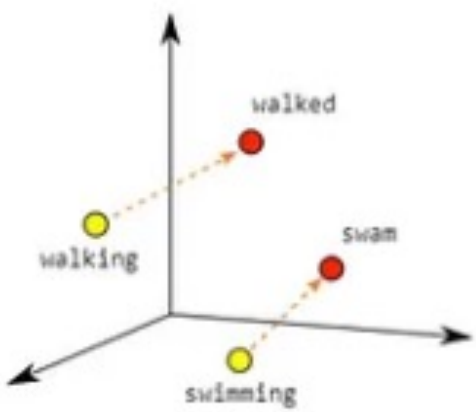
There are two principal representations, both based on constant-length arrays (or vectors).

- One-Hot Encoding

Color
Red
Green
Blue
Green

Red	Green	Blue
1	0	0
0	1	0
0	0	1
0	1	0

- Word Embedding (we'll do these in a few weeks)



We'll generally call sequences/lists/arrays by the standard term vector. For simplicity, we often write them as Python lists here.

Language Modeling: One-Hot Encoding for Words

“one-hot” = only one bit is “hot” at a time

Basic Idea of One-Hot Encoding:

- Create a **vocabulary list** of length N of all distinct words in the text (the ordering is fixed but arbitrary).
- The representation of the k^{th} word in the list is an array/vector of N integers, with a 0 in every position except for a single1 in position k.

Example: “John likes to watch movies. Mary likes movies too.”

Vocabulary list: [“John”, “likes”, “Mary”, ”movies”, “to”, “too”, ”watch”]

“John”	“likes”	“Mary”	”movies”	“to”	“too”	”watch”
0	1	2	3	4	5	6

One-Hot Encodings:

“movies” ⇨ [0, 0, 0, 1, 0, 0, 0]

“likes” ⇨ [0, 1, 0, 0, 0, 0, 0]

Plus:
- vectors have same length;
- spelling is irrelevant.

Minus:
- very long vectors (typically 10,000 or more)

Language Models: Bag-of-Words

Set = unordered collection without duplicates

Bag/Multiset = unordered collection with possible duplicates

Examples of Models: Bag of Words (BOW)

The BOW model represents a text (sentence, sequence of words, entire corpus) as a multiset (bag) of all words in the text, i.e, just the vocabulary, no information about order of words!

Text: "John likes to watch movies. Mary likes movies too."

Vocabulary list:	["John",	"likes",	"Mary",	"movies",	"to",	"too",	"watch"]
	0	1	2	3	4	5	6
BOW model of text:	[1,	2,	1,	2,	1,	1,	1]

Alternate BOW representations:

- We might only consider the presence (0/1) of a word, not its frequency (as if "Set of Words");
- Since most BOW vectors are sparce, we might want to store them as a dictionary:

{ "John" : 1, "likes" : 2, "to" : 1, "watch" : 1, "movies" : 2, "Mary" : 1, "too" : 1 }

Language Models: Bag-of-Words

Question: What is the relationship between One-Hot Encodings and a BOW model?

+

Language Models: Bag-of-Words

Examples of Models: Bag of Words (BOW)

Question: What is the relationship between One-Hot Encodings and a BOW model?

Answer: The BOW model of a text is the array sum of the one-hot encodings:

"John"	[1, 0, 0, 0, 0, 0, 0]
"likes"	[0, 1, 0, 0, 0, 0, 0]
"to"	[0, 0, 0, 0, 1, 0, 0]
"watch"	[0, 0, 0, 0, 0, 0, 1]
"movies"	[0, 0, 0, 1, 0, 0, 0]
"Mary"	[0, 0, 1, 0, 0, 0, 0]
"likes"	[0, 1, 0, 0, 0, 0, 0]
"movies"	[0, 0, 0, 1, 0, 0, 0]
"too"	[0, 0, 0, 0, 0, 1, 0]
+	
	[1, 2, 1, 2, 1, 1, 1]

Language Models: Probabilistic Language Models

Probabilistic Language Model: Assign a probability to text components (letters, words, sentences, ...)

This is very useful to work with the ambiguous nature of human languages, and very amenable to computation:

"*Given K choices for some ambiguous input, choose the most probable one.*"

"I went to they're house on Sunday."

there they're their

Which is most likely?

Language Models: Vector Space Language Models

Vector space models use a vector in M-dimensional space to represent

- Words,
- Sentences, and
- Texts.

This is the current SOTA (“State Of The Art”) for language modeling. Much more on these later!

Probabilistic Language Models

Main Idea of PLMs: Assign a probability to a sequence of words. Why?

Machine Translation:

P(high winds tonight) > P(large winds tonight)

Spelling 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)

Summarization, Q&A, etc.

Probabilistic Language Models

The main task: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

Subtask: compute the conditional probability of the next word

$P(w_5 \mid w_1,w_2,w_3,w_4)$ "The weather is _?_ "

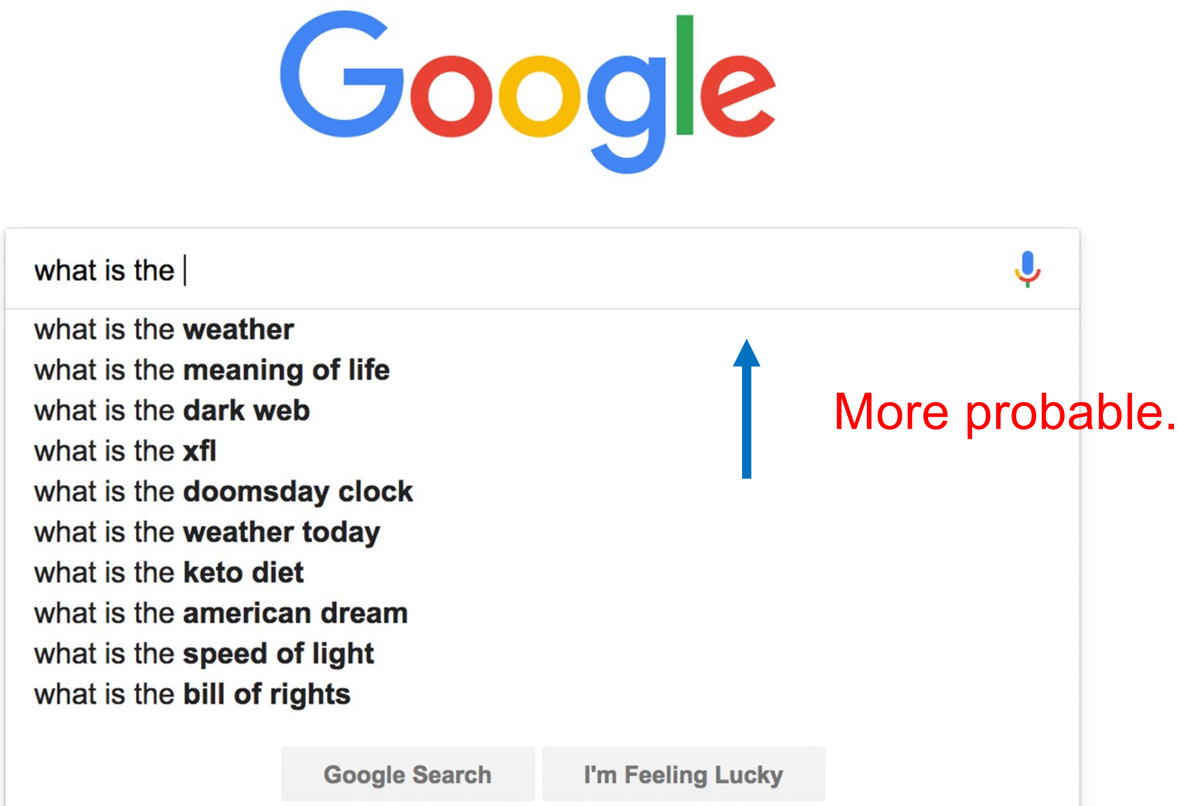
A model that computes either of these:

$P(W)$ or $P(w_n \mid w_1,w_2 \ldots w_{n-1})$

is called a probabilistic language model (often, just "language model").

Probabilistic Language Models

You have seen these before!



Probabilistic Language Modeling

It is possible to apply this framework to any sequence, e.g.,

- Letters in a word; *
- Pitches in a melody;
- Phonemes in a voice signal;
- Sentences in a paragraph; or
- Topics in a discourse.

* This was actually the first use of this model by Markov (1913) as an example of the new concept of Markov Chains; also used by Shannon (1948) in his foundational paper on Information Theory.   It is possible to apply this model to spell check (a good project!).

Probabilistic Language Modeling

How to compute P(W)?

How to compute this joint probability:

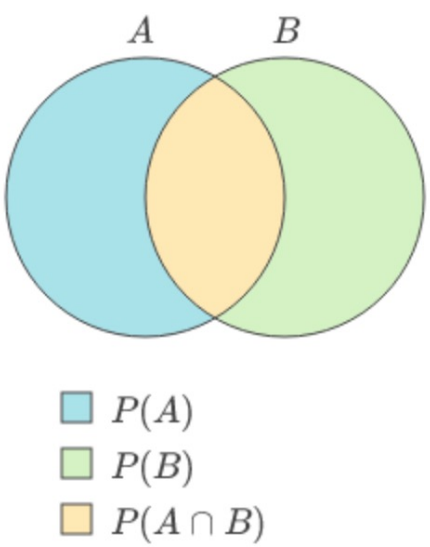
P(I went to their house on Sunday)

Intuition: let's rely on the Chain Rule of Probability

Calculating Probabilities

Recall the definition of conditional probabilities

p(A|B) = P(B,A) / P(B) Rewriting: P(B,A) = P(B) * P(A|B)



Conditional Probability Formula

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability that A occurs given that B has already occured

For this LM, we will think of B,A as a sequence:

B happens, then A happens.

Thus, P(A|B) = “given that B has happened, what is the probablity that A happens”?

Calculating Probabilities

- Recall the definition of conditional probabilities

  $p(B|A) = P(A,B) / P(A)$ Rewriting: $P(A,B) = P(A) * P(B|A)$

- More variables:

  $P(A,B,C,D) = P(A) \times P(B|A) \times P(C|A,B) \times P(D|A,B,C)$

- General Chain Rule:
  - $P(x_1,x_2,x_3,\ldots,x_n) = P(x_1) \times P(x_2|x_1) \times P(x_3|x_1,x_2) \times \ldots \times P(x_n|x_1,\ldots,x_{n-1})$

Calculating Probabilities

The Chain Rule applied to compute joint probability of words in sentence:

"I went to their house on Sunday."

$$P(w_1 w_2 \ldots w_n) = \prod_{1 \le i \le n} P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P(I went to their house on Sunday.) =

P(I) × P(went | I) × P(to | I went) × P(their | I went to)

× P(house | I went to their) × P(on | I went to their house)

x P(Sunday | I went to their house on)

Calculating Probabilities

How to estimate these probabilities?

Could we just count and divide?

$$P(\text{ Sunday } | \text{ I went to their house on }) = \frac{\text{Count(I went to their house on Sunday)}}{\text{Count(I went to their house on)}}$$

Calculating Probabilities

Can we just count? Not realistic:

In an infinite set of sentences, the probability of any distinct sequence of words is 0. So a data set is a small sample which hopefully represents the essential features of the language.

But realistic data sets never have enough sample sequences, and sequences might be very long or simply not exist in your data.

"I have been here before," I said; I had been there before; first with Sebastian more than twenty years ago on a cloudless day in June, when the ditches were white with fools' parsley and meadowsweet and the air heavy with all the scents of summer; it was a day of peculiar splendor, such as our climate affords once or twice a year, when leaf and flower and bird and sun-lit stone and shadow seem all to proclaim the glory of God; and though I had been there so often, in so many moods, it was to that first visit that my heart returned on this, my latest." Evelyn Waugh: *Brideshead Revisited, first sentence*.

Calculating Probabilities

Markov Assumption: Finite history!

Only consider N-1 words

of left context, for some fixed N.

Andrei Markov

So, if N = 2

I went to their house on Sunday

I went

went to

to their

their house

house on

on Sunday

Terminology: An N-Gram is a sequence of N contiguous words from the data set.

unigram = 1-gram,
bigram = 2-gram,
trigram = 3-gram, etc.

Calculating Probabilities

Andrei Markov

Markov Assumption:

Only consider N-1 words

of left context, for some fixed N.

If N = 3

I went to their house on Sunday

I went to

went to their

to their house

their house on

house on Sunday

Calculating Probabilities

Markov Assumption: Only consider N-1 words of left context.

Thus, for a sequence of length M,

$$P(w_1 w_2 \ldots w_M) \approx \prod_{N \leq i \leq M-N} P(w_i | w_{i-N+1} \ldots w_{i-1})$$

Calculating Probabilities

Bigram Example (N = 2)

P(<s> I went to their house on Sunday </s>)=

P(I |<s>) × P(went | I) × P(to | went) × P(their | to)

× P(house | their) × P(on | house)

x P(Sunday | on) x P(</s> | Sunday)

$$P(\,\mathrm{I} \mid \text{<s>}\,) \approx \frac{C(\,\text{<s>}\ \mathrm{I}\,)}{C(\,\text{<s>}\,)}$$

$$P(\,\text{went} \mid \mathrm{I}\,) \approx \frac{C(\,\mathrm{I}\ \text{went}\,)}{C(\,\mathrm{I}\,)}$$

Note that this calculation involves finding the number of occurrences of an N-gram and of an (N-1)-gram (the prefix)!

Calculating Probabilities

Trigram Example (N = 3)

$$P(\text{<s> I went to their house on Sunday </s>}) =$$

$$P(\text{I} \mid \text{<s>}) \times P(\text{went} \mid \text{I}) \times P(\text{to} \mid \text{went}) \times P(\text{their} \mid \text{to})$$

$$\times\ P(\text{house} \mid \text{their}) \times P(\text{on} \mid \text{house})$$

$$\text{x}\ P(\text{Sunday} \mid \text{on})\ \text{x}\ P(\text{</s>} \mid \text{Sunday})$$

$$P(\text{went} \mid \text{<s> I}) = \frac{C(\text{<s> I went})}{C(\text{<s> I})}$$

$$P(\text{to} \mid \text{I went}) = \frac{C(\text{I went to})}{C(\text{I went})}$$

Calculating Probabilities

Remarks:

This is almost trivial to code after you have separated your text into words and sentences.

For small N, it will be reasonably efficient.

BUT, it does NOT capture the recursive/nesting structure inherent in complex sentences:

My friend Bill, who went to the same high school that I did –Pennsbury, which is in Fairless Hills in PA—lives in his car, and he called me yesterday (or the day before, I forget).

Calculating Probabilities

An example

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

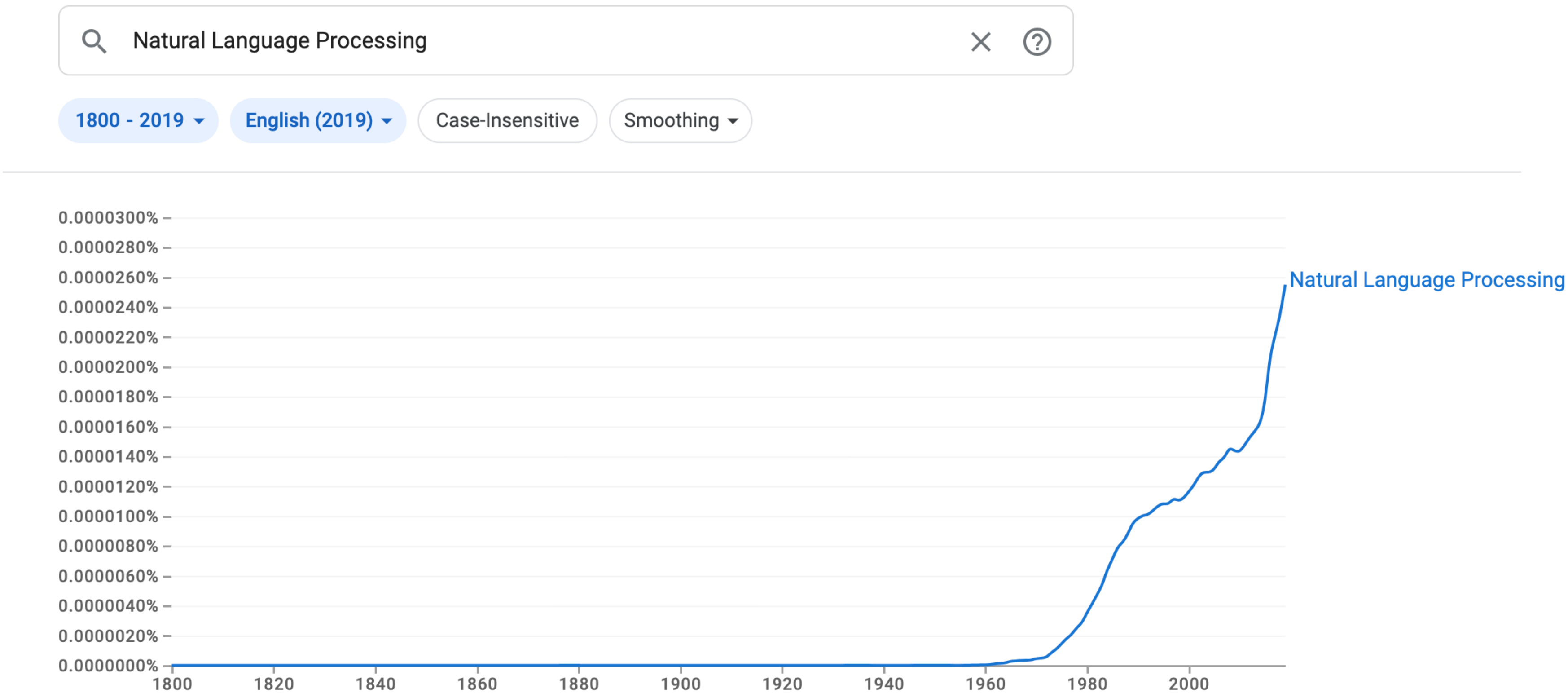
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

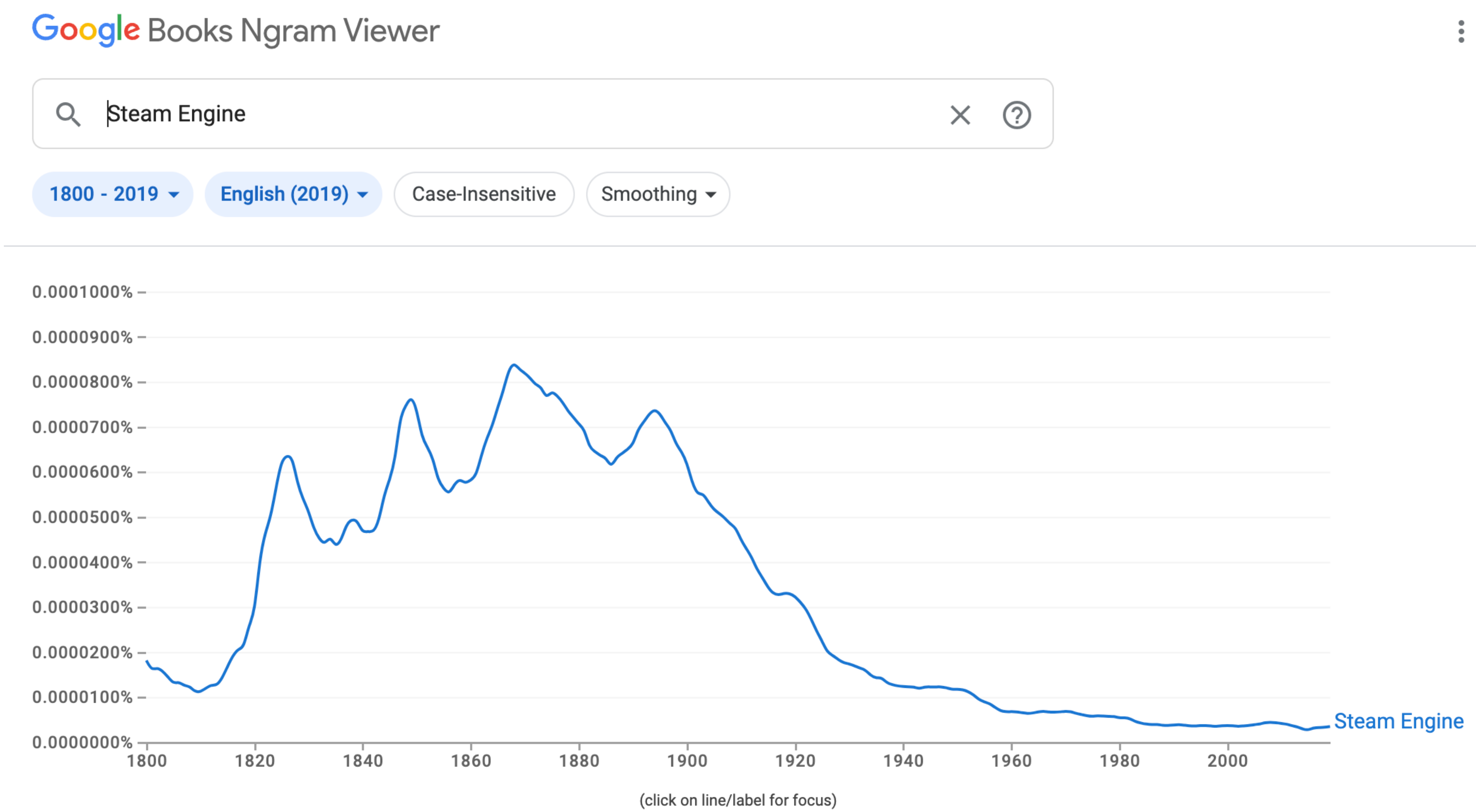
Google Book N-grams

- https://books.google.com/ngrams



Google Book N-grams

- https://books.google.com/ngrams



Generative Language Models

A clever feature of this model is that it can easily generate sentences.

For bigrams, after calculating the probability of all bigrams appearing in the data.

Pick a probable* bigram <s> w_1
Pick a probable bigram w_1 w_2
.... etc. ...
End when you generate a bigram w_k </s>

* You may not want to always choose the most likely, or you will not be able to generate many sentences! So choose randomly from the probability distribution of next words.

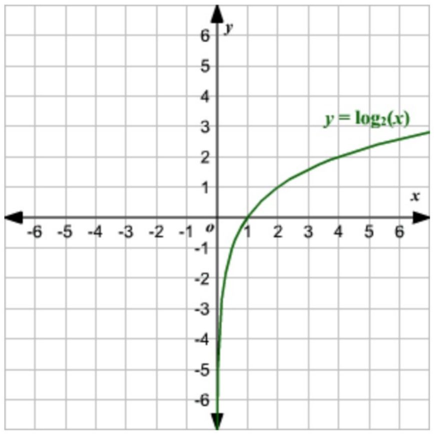
Optional: An Important Practical Issue:

- **We do everything in log space!**
  - Avoid loss of precision from underflow (prob p might be tiny)
  - Adding is *much* faster than multiplying
  - log is monotonic, so it preserves order

probs ($p \geq 0$):

$$p < q \;\leftrightarrow\; \log(p) < \log(q)$$

  - Can easily recover probs using exp(...)



$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$