Language Modeling

David Yarowsky

10/14/2019

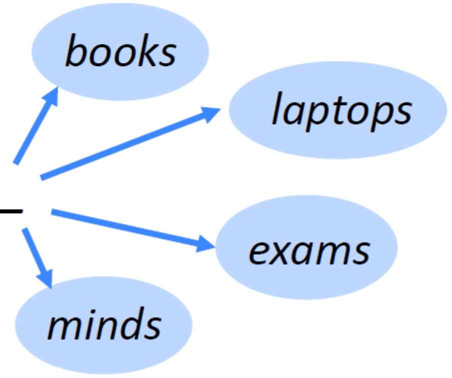
Acknowledgements and thanks to:

- Michael Doroshenko
- Alexey Karyakin
- Dan Jurafsky
- Jason Eisner
- Kai-Wei Chang

Language Modeling

- Language Modeling is the task of predicting what word comes next.

the students opened their ______



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a Language Model.

Language Modeling

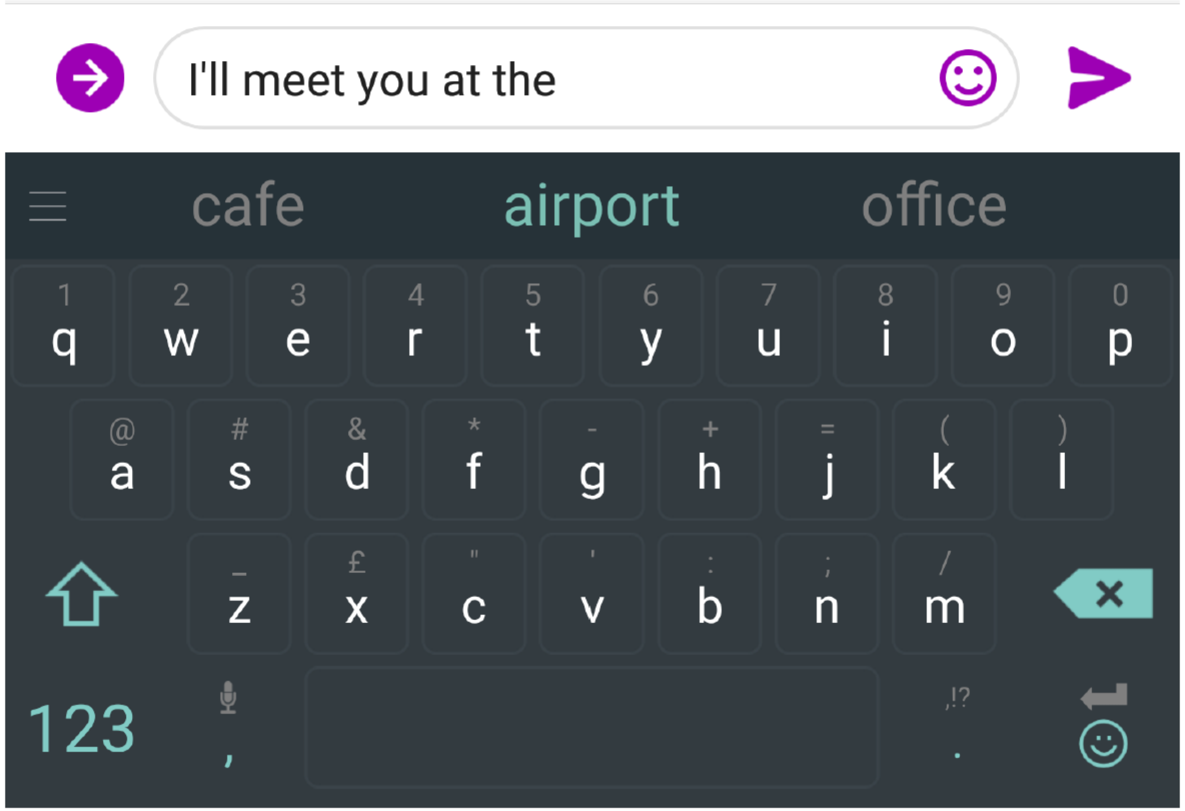
- You can also think of a Language Model as a system that assigns probability to a piece of text.

- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

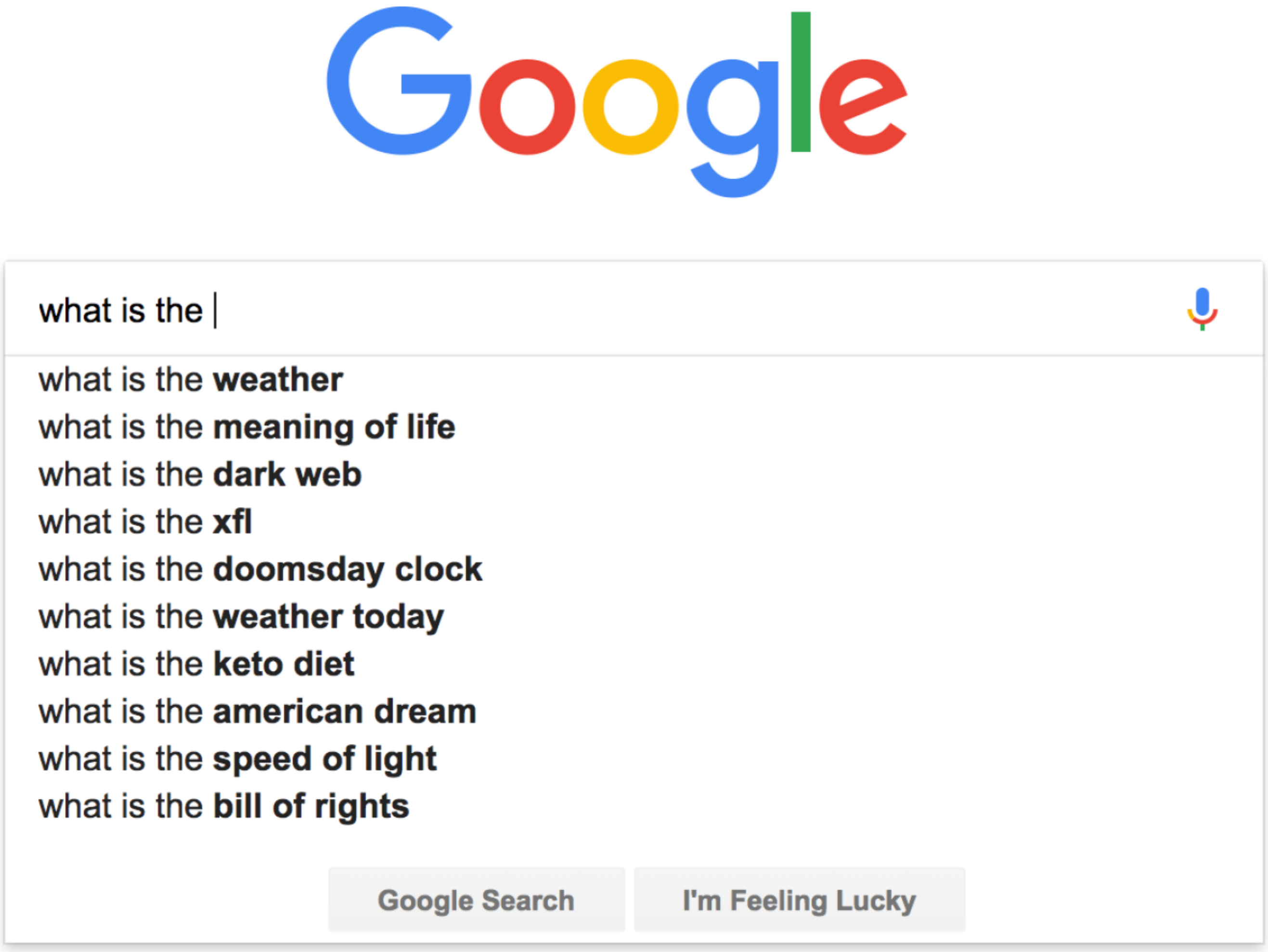
$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}}$$

You use Language Models every day!

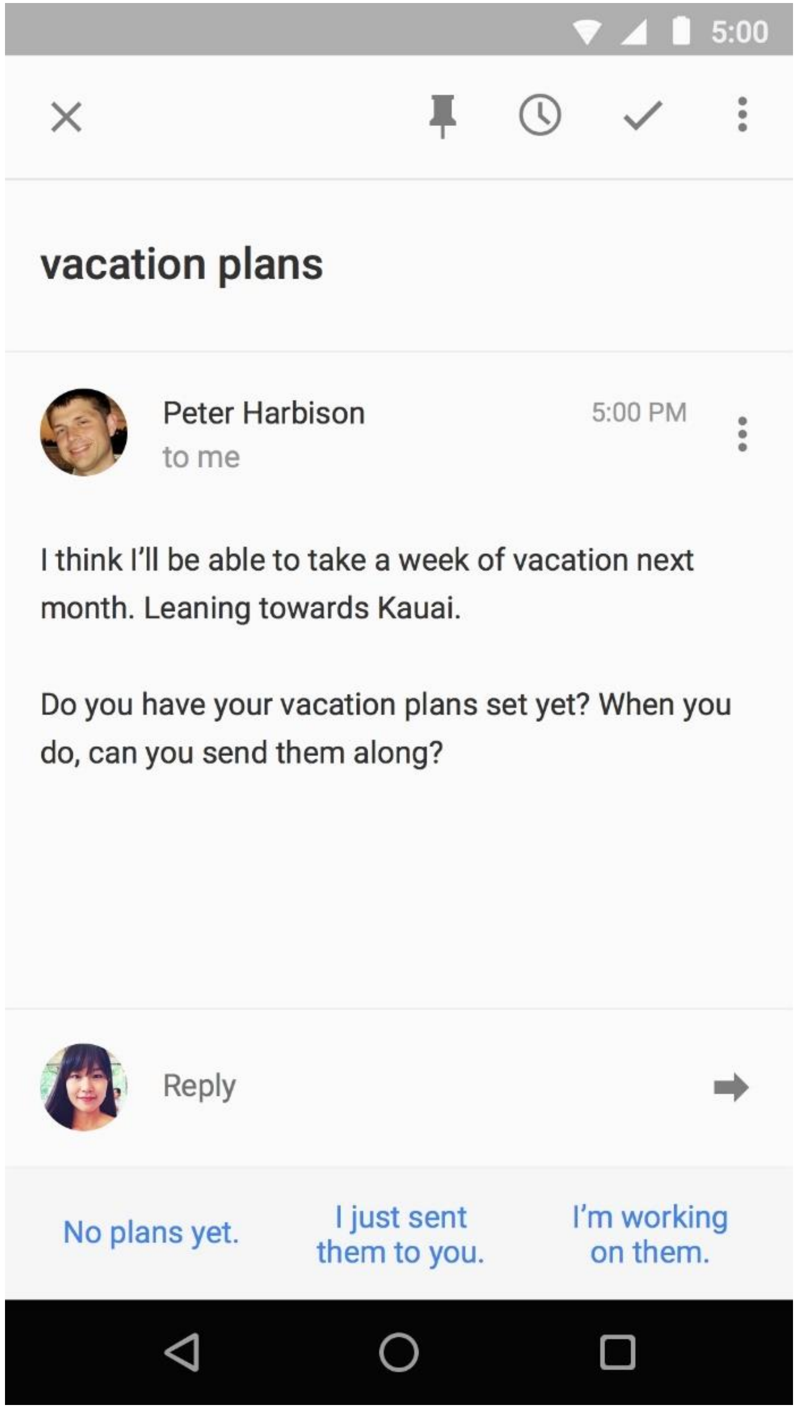


You use Language Models every day!



You use language modeling every day

Smart Reply



Infamous uses of Language Modeling

Language generation

https://pdos.csail.mit.edu/archive/scigen/

Deploying Superblocks and Compilers

Julia and Dan

Abstract

Recent advances in replicated algorithms and relational symmetries have paved the way for architecture. After years of natural research into erasure coding, we show the deployment of courseware, which embodies the key principles of steganography. *Loy*, our new system for the exploration of sensor networks, is the solution to all of these issues.

1 Introduction

Steganographers agree that robust symmetries are an interesting new topic in the field of cryptography, and information theorists concur. We view operat-

thesize unstable algorithms, we fulfil
without investigating the evaluation of

Our contributions are threefold. Firs
how erasure coding can be applied to
tion of reinforcement learning. We p
algorithm for the deployment of extr
ming (*Loy*), which we use to prove tha
and operating systems [19, 7, 14] can
fill this goal. we examine how replic
be applied to the deployment of linked

The rest of this paper is organized a
marily, we motivate the need for fibe
We demonstrate the synthesis of the Tu
Finally, we conclude.

Why should we care about Language Modeling?

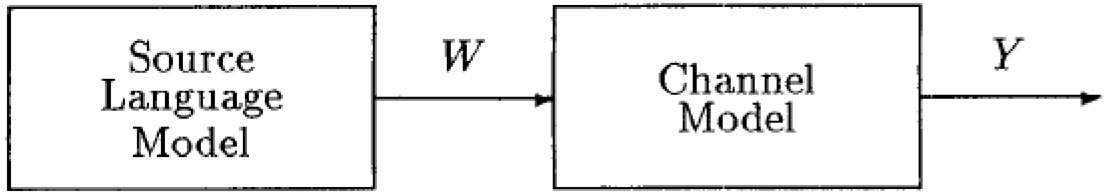
- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
    - Predictive typing
    - Speech recognition
    - Handwriting recognition
    - Spelling/grammar correction
    - Authorship identification
    - Machine translation
    - Summarization
    - Dialogue
    - etc.

Traditional channel model applications of LM's

Application	Signal Y
automatic speech recognition	acoustic signal
machine translation	sequence of words in a foreign language
spelling correction	sequence of characters produced by a possibly imperfect typist

source-channel model



$$\Pr(W) \quad \times \quad \Pr(Y \mid W) = \Pr(W, Y)$$

The ultimate goal is to determine W from Y

Traditional channel model applications of LM's

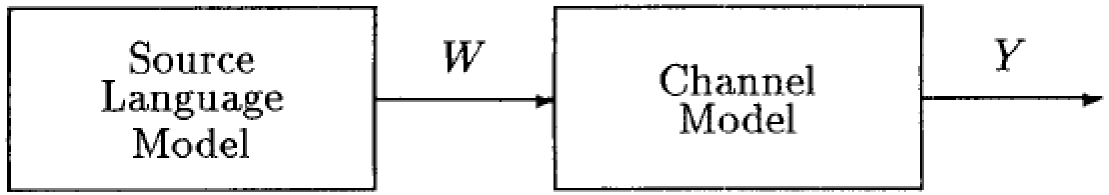
Speech Recognition:

- How do you recognize speech?
- How do you wreck a nice beach?

OCR/Handwriting Recognition:

- Federal farm aid
- Federal form aid

source-channel model



$$\Pr(W) \times \Pr(Y \mid W) = \Pr(W, Y)$$

The ultimate goal is to determine W from Y

Traditional channel model applications of LM's

Machine Translation:

- Choose randomly among outputs:
  - Visitant which came into the place where it will be Japanese has admired that there was Mount Fuji.
- Top 10 outputs according to bigram probabilities:
  - Visitors who came in Japan admire Mount Fuji.
  - Visitors who came in Japan admires Mount Fuji.
  - Visitors who arrived in Japan admire Mount Fuji.
  - Visitors who arrived in Japan admires Mount Fuji.
  - Visitors who came to Japan admire Mount Fuji.
  - A visitor who came in Japan admire Mount Fuji.
  - The visitor who came in Japan admire Mount Fuji.
  - Visitors who came in Japan admire Mount Fuji.
  - The visitor who came in Japan admires Mount Fuji.
  - Mount Fuji is admired by a visitor who came in Japan.

- Automatic Yahoo classification, etc.
- Similar to language ID …
  - Topic 1 sample: In the beginning God created …
  - Topic 2 sample: The history of all hitherto existing society is the history of class struggles. …
- Input text: Matt's Communist Homepage. Capitalism is unfair and has been ruining the lives of millions of people around the world. The profits from the workers' labor …
- Input text: And they have beat their swords to ploughshares, And their spears to pruning-hooks. Nation doth not lift up sword unto nation, neither do they learn war any more. …

Some History

- Chomsky (in *Syntactic Structures* (1957)):

> Second, the notion "grammatical" cannot be identified with "meaningful" or "significant" in any semantic sense. Sentences (1) and (2) are equally nonsensical, but any speaker of English will recognize that only the former is grammatical.
>
> (1) Colorless green ideas sleep furiously.
>
> (2) Furiously sleep ideas green colorless.
>
> . . .
>
> . . . Third, the notion "grammatical in English" cannot be identified in any way with the notion "high order of statistical approximation to English". It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out on identical grounds as equally 'remote' from English. Yet (1), though nonsensical, is grammatical, while (2) is not. . . .

(my emphasis)

n-gram Language Models

the students opened their ______

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn a *n*-gram Language Model!

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

n-gram Language Models

- First we make a simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *n-1* words.

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}$$

(definition of conditional prob)

- **<u>Question:</u>** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **<u>Answer:</u>** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* ______

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
  - → P(books | students opened their) = 0.4
- “students opened their exams” occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the “proctor” context?

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their $\boldsymbol{w}$"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

(Partial) Solution: Add small δ to the count for every $\boldsymbol{w} \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\boxed{\text{count}(\text{students opened their})}}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse.* Typically we can't have *n* bigger than 5.

Storage Problems with n-gram Language Models

<u>Storage</u>: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the ________

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

*** Try for yourself:** https://nlpforhackers.io/language-models/

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

$\underbrace{\textit{today the}}_{\text{condition on this}}$ ________

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

sample

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price ________

condition on this

get probability distribution

of	0.308	sample
for	0.050	
it	0.046	
to	0.046	
is	0.031	
…		

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price of ________

condition on this

get probability distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

sample

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price of gold ________

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size…

Probabilistic Language Models

- The goal: assign a probability to a sentence
  - Machine Translation:
    - P(**high** winds tonite) > P(**large** winds tonite)
  - 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, question-answering, etc., etc.!!

Probabilistic Language Modeling

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

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

- Related task: probability of an upcoming word:

  $P(w_5|w_1,w_2,w_3,w_4)$

- A model that computes either of these:

  $P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$ is called a **language model**.

- Better: **the grammar** But **language model** or **LM** is standard

Evaluation and Perplexity

Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to "real" or "frequently observed" sentences
    - Than "ungrammatical" or "rarely observed" sentences?
- We train parameters of our model on a **training set**.
- We test the model's performance on data we haven't seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, machine translation system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So instead
  - Sometimes use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
  - But is helpful to think about.

Intuition of Perplexity

- How well can we predict the next word?

  I always order pizza with cheese and ____

  The 33rd President of the US was ____

  I saw a ____

  - Unigrams are terrible at this game. (Why?)
- A better model
  - is one which assigns a higher probability to the word that actually occurs

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Chain rule:

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

For bigrams:

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

Minimizing perplexity is the same as maximizing probability

Example

- How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
  - Perplexity 10
- How hard is recognizing (30,000) names at Microsoft.
  - Perplexity = 30,000
- If a system has to recognize
  - Operator (25% of the time)
  - Sales (25% of the time)
  - Technical Support (25% of the time)
  - 30,000 names (overall 25% of the time, 1 in 120,000 each)
  - Perplexity is 52.64 ≈ 53 – computed via the geometric mean formula
- Perplexity is weighted equivalent branching factor (number of possible children)

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

QUESTION 3

A traffic signal has three colors: green, yellow, and red, which appear with the following probabilities. Using a unigram model, **what is the perplexity of the sequence (green, yellow, red)?**

P(green) = 2/5

P(yellow) = 1/5

P(red) = 2/5

$$PP(green, yellow, red) = \left(\frac{2}{5} \times \frac{1}{5} \times \frac{2}{5}\right)^{-\frac{1}{3}}$$

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Generalization and zeros

The Shannon Visualization Method

- Choose a random bigram

  (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

N-gram approximations to Shakespeare

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

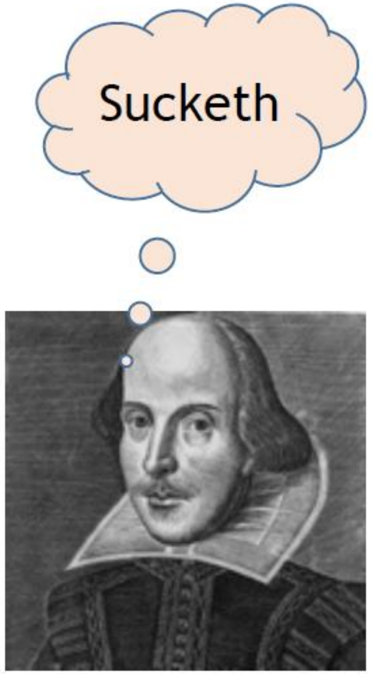
What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.



Shakespeare as corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams.
  - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

The Wall Street Journal is not Shakespeare

Unigram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

Bigram

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

Trigram

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

The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
      - But occur in the test set

	i	want	to	eat	chinese
i	0.002	0.33	0	0.0036	0
want	0.0022	0	0.66	0.0011	0.0065
to	0.00083	0	0.0017	0.28	0.00083
eat	0	0	0.0027	0	0.021
chinese	0.0063	0	0	0	0
food	0.014	0	0.014	0	0.00092
lunch	0.0059	0	0	0	0
spend	0.0036	0	0.0036	0	0

Zeros

- Training set:

  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

  P("offer" | denied the) = 0

- Test set

  … denied the offer
  … denied the loan

Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can’t divide by 0)!

- Zero mitigation
  - Various **smoothing** techniques

Basic Smoothing: Interpolation and Back-off

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram

- Interpolation works better

Linear Interpolation

- Simple interpolation

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1})P(w_n)$$

QUESTION 4

Suppose we train unigram, bigram and trigram language models on the following corpus:

<s> I am Sam </s>

<s> Sam I am </s>

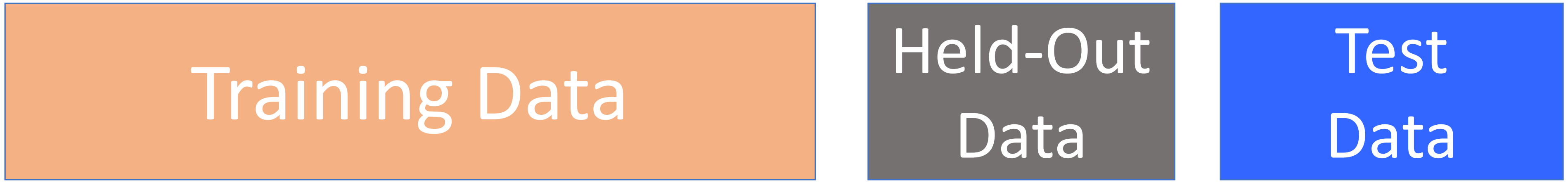
<s> I do not like green eggs and ham </s>

What is P(Sam|I am) if we use linear interpolation with λi=13?

$$P(Sam|I\ am) = \frac{1}{3}P(Sam) + \frac{1}{3}P(Sam|am) + \frac{1}{3}P(Sam|I\ am) =$$
$$\frac{1}{3} \times \frac{2}{20} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times \frac{1}{2}$$

How to set the lambdas?

- Use a **held-out** corpus



- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Unknown words: open vs closed vocabulary

- If we know all the words in advance
  - Vocabulary V is fixed
  - Closed vocabulary task
- Often we don’t know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token **<UNK>**
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold
    - Remove singletons of higher-order n-grams
  - Entropy-based pruning

Back-off: Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
  - works well at large scale
- Use MLE or back-off to a lesser order n-gram
- Does not produce probability, but scores

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Uses of Language Models

- Speech recognition
  - “I ate a cherry” is a more likely sentence than “Eye eight uh Jerry”
- OCR & Handwriting recognition
  - More probable sentences are more likely correct readings.
- Machine translation
  - More likely sentences are probably better translations.
- Generation
  - More likely sentences are probably better NL generations.
- Context sensitive spelling correction
  - “Their are problems wit this sentence.”

Completion Prediction

- A language model also supports predicting the completion of a sentence.
  - Please turn off your cell _____
  - Your program does not ______
- *Predictive text input* systems can guess what you are typing and give choices on how to complete it.

N-Gram Models

- Estimate probability of each word given prior context.
  - P(phone | Please turn off your cell)
- Number of parameters required grows exponentially with the number of words of prior context.
- An N-gram model uses only N–1 words of prior context.
  - Unigram: P(phone)
  - Bigram: P(phone | cell)
  - Trigram: P(phone | your cell)
- The ***Markov assumption*** is the presumption that the future behavior of a dynamical system only depends on its recent history. In particular, in a ***kth-order Markov model***, the next state only depends on the *k* most recent states, therefore an N-gram model is a (N–1)-order Markov model.

N-Gram Model Formulas

- Word sequences

$$w_1^n = w_1 \ldots w_n$$

- Chain rule of probability

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2) \ldots P(w_n \mid w_1^{n-1}) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

- Bigram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

- N-gram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-N+1}^{k-1})$$

Estimating Probabilities

- N-gram conditional probabilities can be estimated from raw text based on the ***relative frequency*** of word sequences.

Bigram: $$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

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

- To have a consistent probabilistic model, append a unique start (<s>) and end (</s>) symbol to every sentence and treat these as additional words.

Smoothing

- Since there are a combinatorial number of possible word sequences, many rare (but not impossible) combinations never occur in training, so MLE incorrectly assigns zero to many parameters (a.k.a. ***sparse data***).
- If a new combination occurs during testing, it is given a probability of zero and the entire sequence gets a probability of zero (i.e. infinite perplexity).
- In practice, parameters are ***smoothed*** (a.k.a. ***regularized***) to reassign some probability mass to unseen events.
  - Adding probability mass to unseen events requires removing it from seen ones (***discounting***) in order to maintain a joint distribution that sums to 1.

Laplace (Add-One) Smoothing

- "Hallucinate" additional training data in which each possible N-gram occurs exactly once and adjust estimates accordingly.

Bigram: $$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

N-gram: $$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)+1}{C(w_{n-N+1}^{n-1})+V}$$

where *V* is the total number of possible (N−1)-grams (i.e. the vocabulary size for a bigram model).

- Tends to reassign too much mass to unseen events, so can be adjusted to add 0<δ<1 (normalized by δ*V* instead of *V*).

Advanced Smoothing

- Many advanced techniques have been developed to improve smoothing for language models.
  - Good-Turing
  - Interpolation
  - Backoff
  - Kneser-Ney
  - Class-based (cluster) N-grams

Model Combination

- As N increases, the power (expressiveness) of an N-gram model increases, ***but*** the ability to estimate accurate parameters from sparse data decreases (i.e. the smoothing problem gets worse).
- A general approach is to combine the results of multiple N-gram models of increasing complexity (i.e. increasing N).

Interpolation

- Linearly combine estimates of N-gram models of increasing order.

Interpolated Trigram Model:

$$\hat{P}(w_n \mid w_{n-2,} w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2,} w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n)$$

Where: $\sum_i \lambda_i = 1$

- Learn proper values for λ_i by training to (approximately) maximize the likelihood of an independent ***development*** (a.k.a. ***tuning***) corpus.

Backoff

- Only use lower-order model when data for higher-order model is unavailable (i.e. count is zero).
- Recursively back-off to weaker models until data is available.

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^{n}) > 1 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

Where P* is a discounted probability estimate to reserve mass for unseen events and α's are back-off weights (see text for details).

Practical Issues

- We do everything in the log space
  - Avoid underflow
  - Adding is faster than multiplying

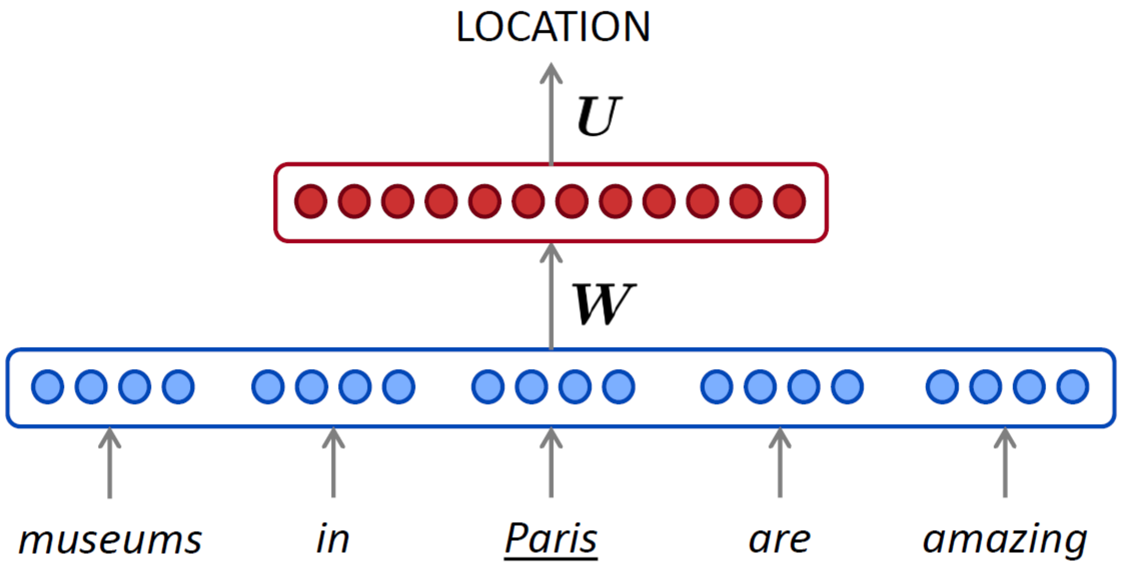
$$\log(p_1 \times p_2) = \log(p_1) + \log(p_2)$$

- Toolkits
  - KenLM: https://kheafield.com/code/kenlm/
  - SRILM: http://www.speech.sri.com/projects/srilm

How to build a *neural* Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?
  - We saw this applied to Named Entity Recognition in Lecture 3:



A fixed-window neural Language Model

~~*as the proctor started the clock*~~ $\underbrace{\textit{the students opened their}}_{\text{fixed window}}$ ______

discard

A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

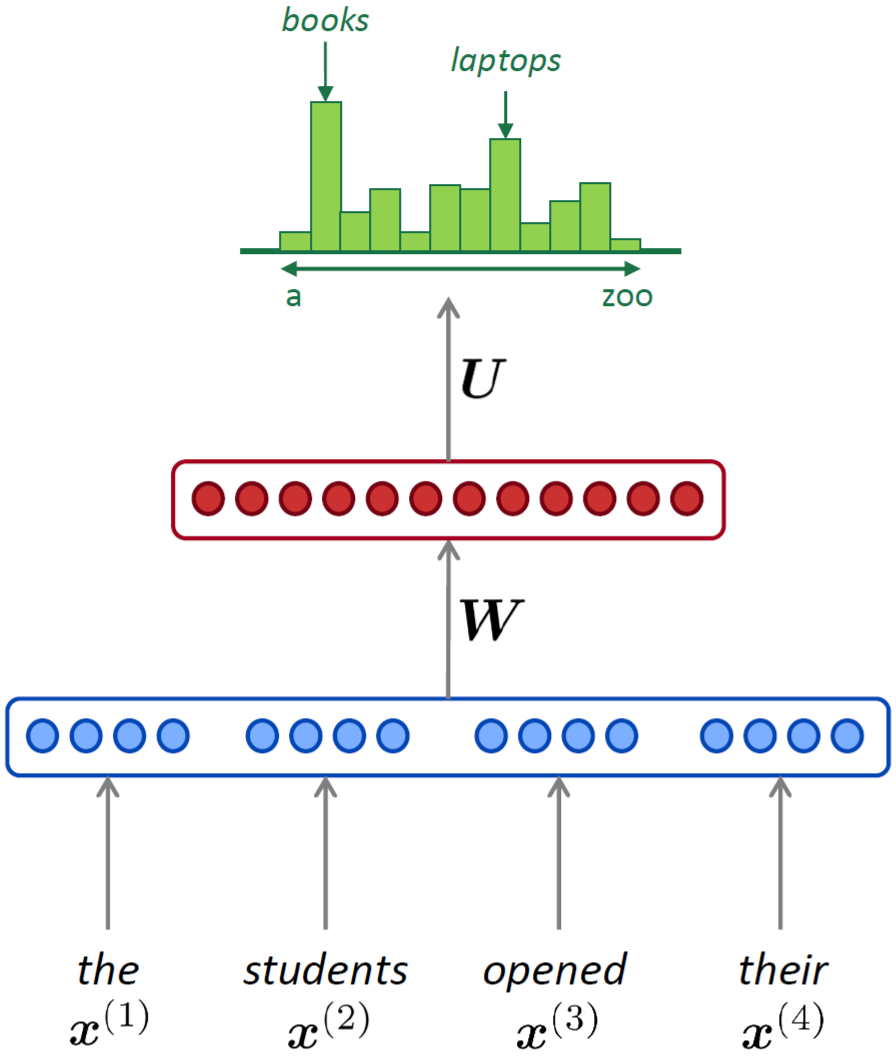
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



A fixed-window neural Language Model

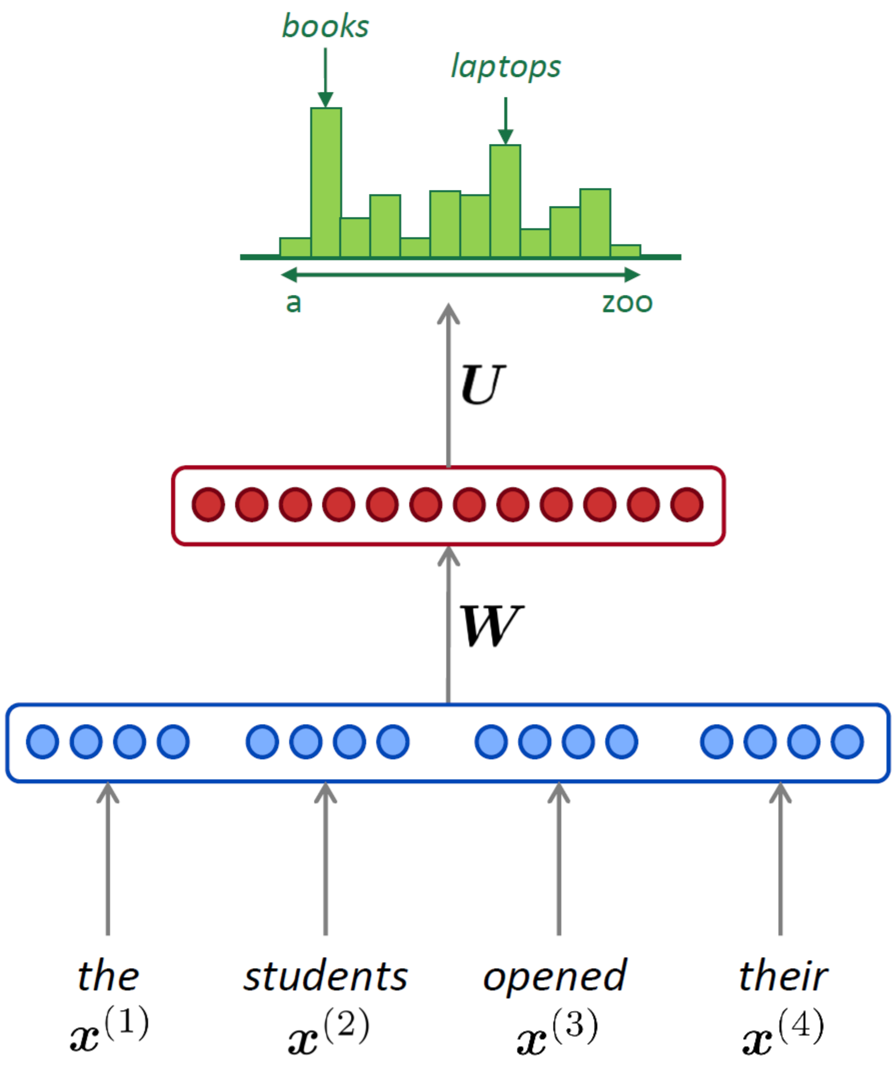
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

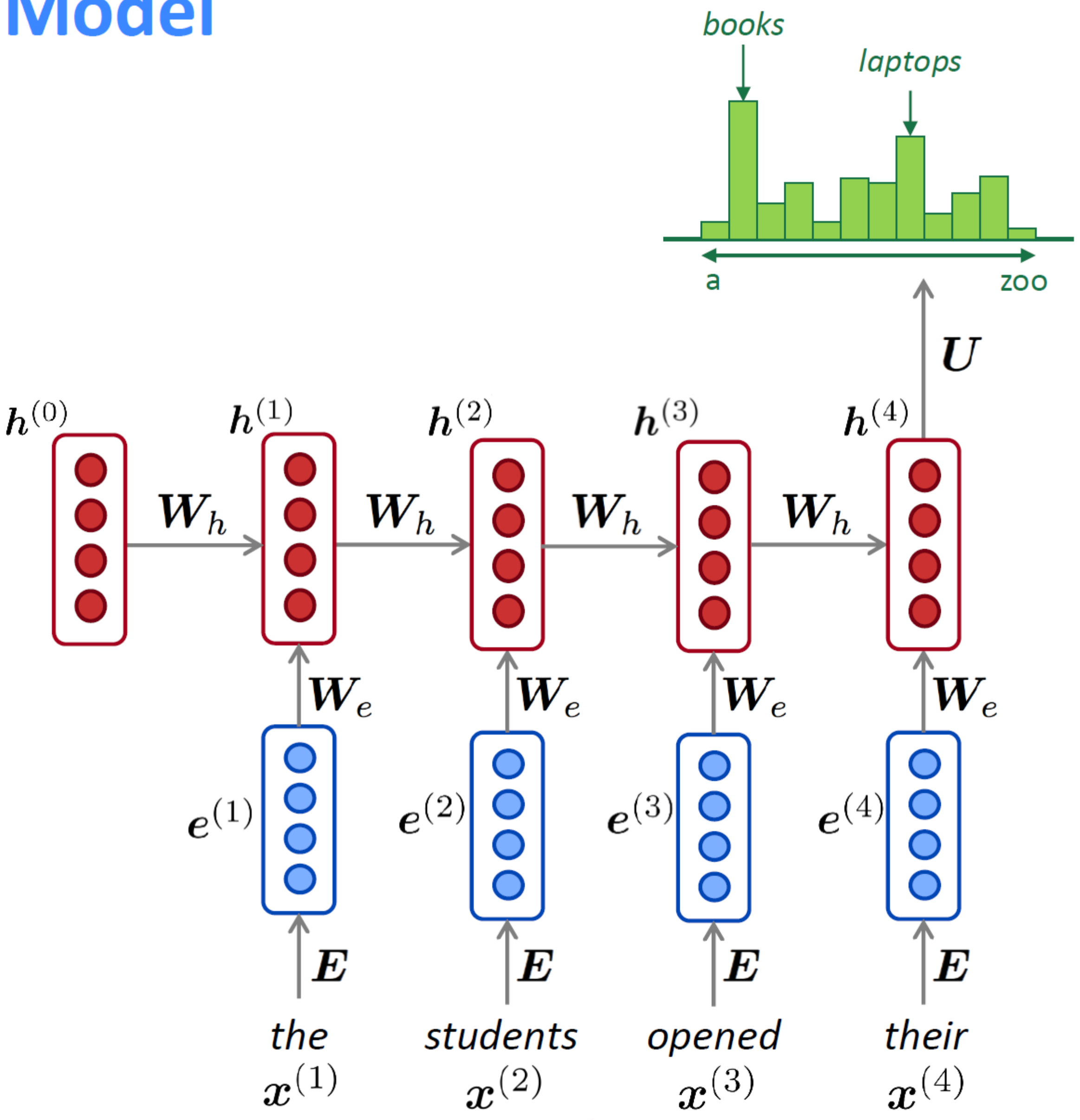
- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



A RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

***Note**: this input sequence could be much longer, but this slide doesn't have space!*

A RNN Language Model

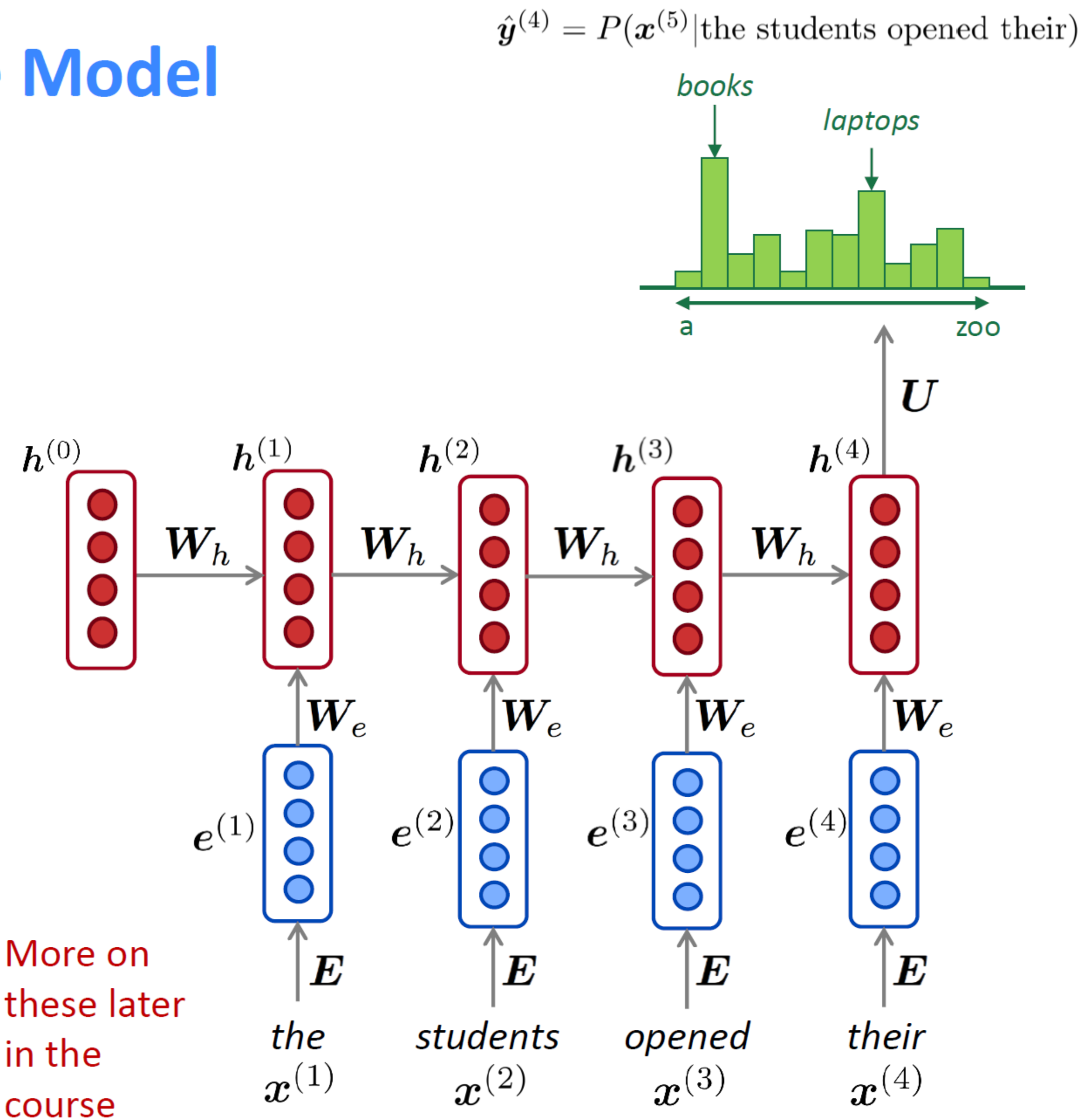
RNN **Advantages**:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course



Generating text with a RNN Language Model

- Let’s have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN Language Model

- Let’s have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

“Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Generating text with a RNN Language Model

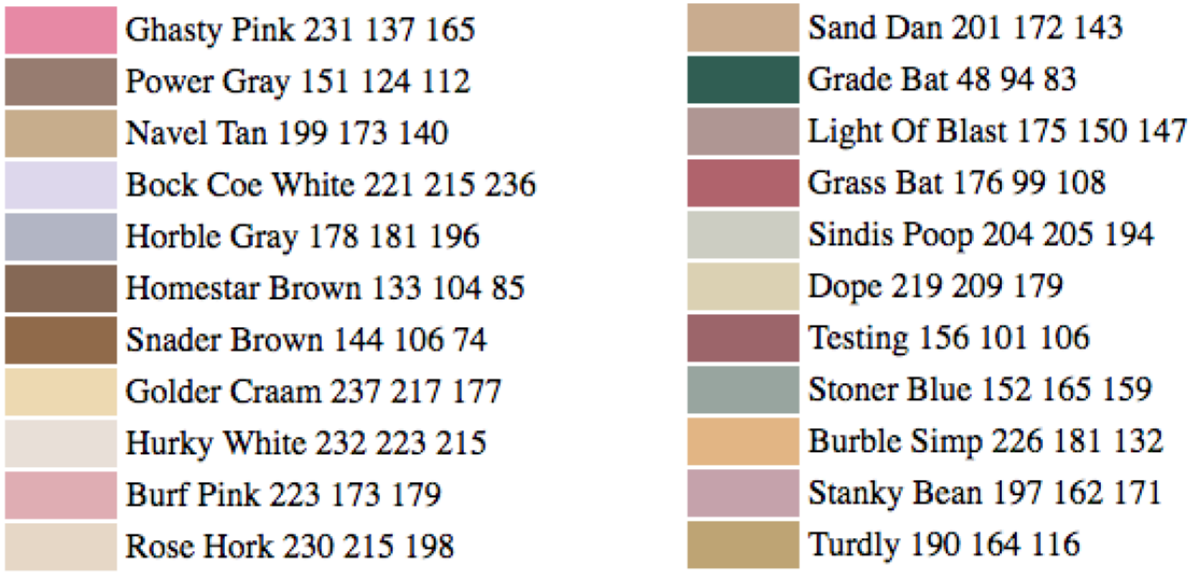
- Let’s have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on recipes:

```
    Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
    Yield: 6 Servings

    2 tb Parmesan cheese -- chopped
    1 c  Coconut milk
    3    Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on paint color names:



This is an example of a character-level RNN-LM (predicts what character comes next)

Source: http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

RNNs have greatly improved perplexity

n-gram model

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Resources:

- Google n-gram:

https://research.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens: 1,024,908,267,229
Number of sentences: 95,119,665,584
Number of unigrams: 13,588,391
Number of bigrams: 314,843,401
Number of trigrams: 977,069,902
Number of fourgrams: 1,313,818,354
Number of fivegrams: 1,176,470,663
```

- 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
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

More resources

- Google n-gram viewer

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

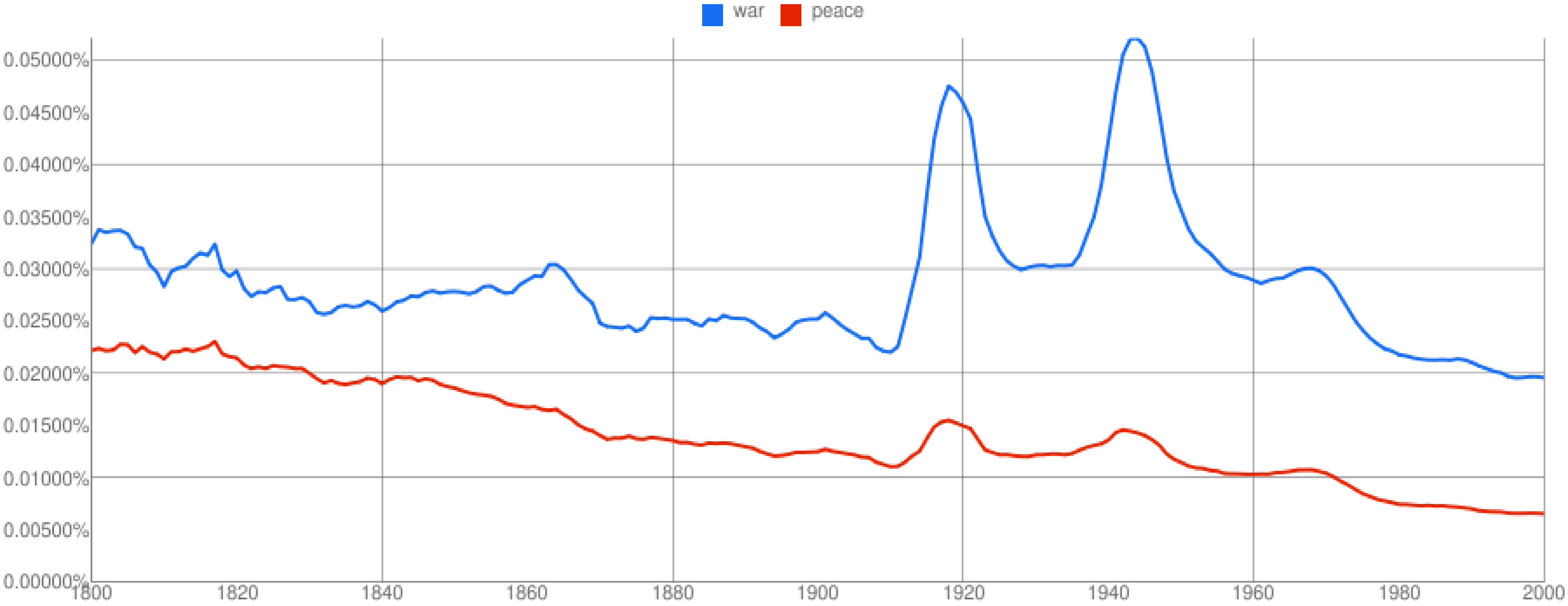
Data:
http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

circumvallate 1978 335 91
circumvallate 1979 261 91

Graph these **case-sensitive** comma-separated phrases: war,peace

between 1800 and 2000 from the corpus English with smoothing of 3 .

Search lots of books

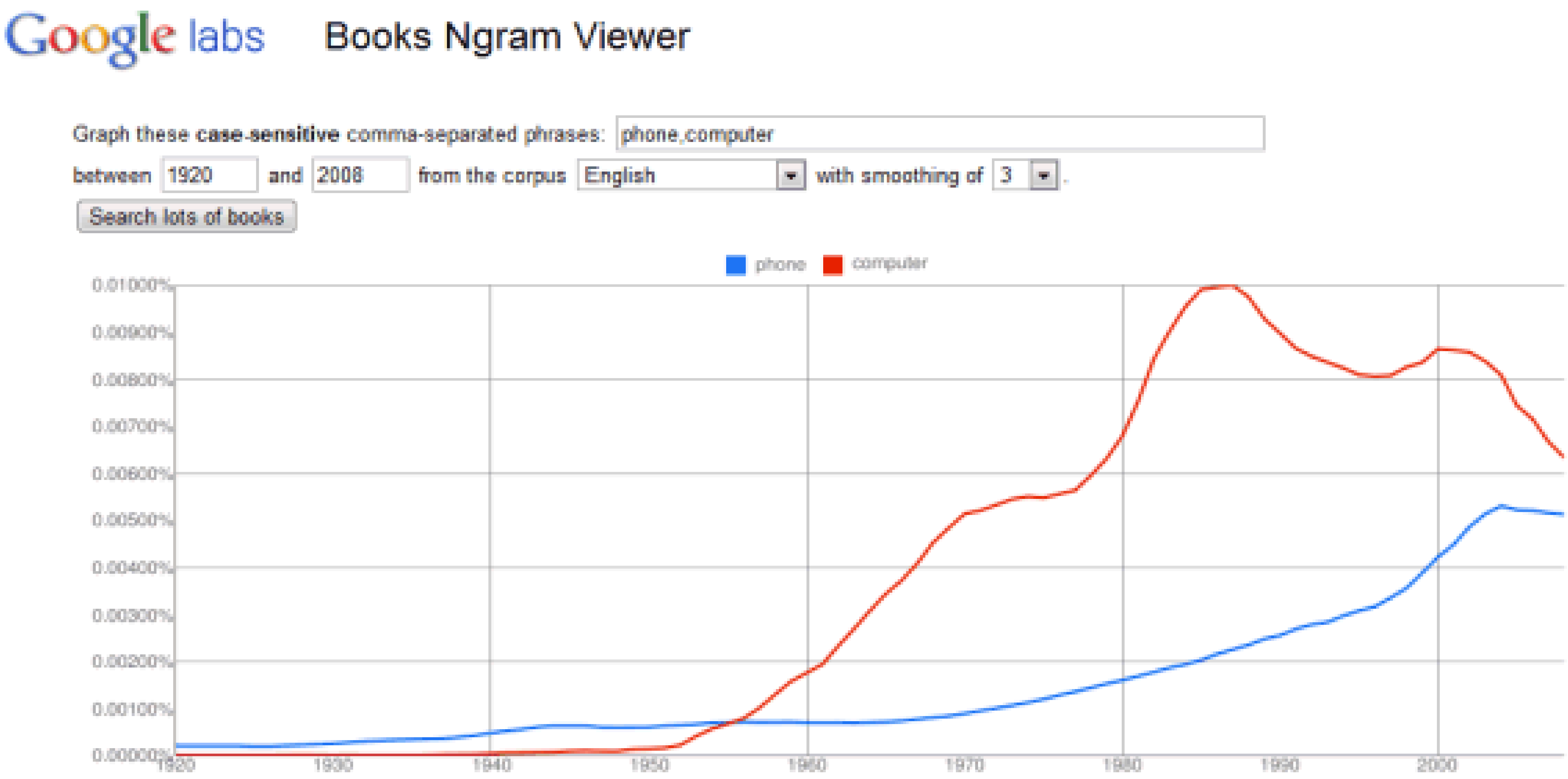


Search in Google Books:

1800 - 1817	1818 - 1939	1940 - 1952	1953 - 1971	1972 - 2000	war (English)
1800 - 1812	1813 - 1825	1826 - 1872	1873 - 1963	1964 - 2000	peace (English)

A Problem for N-Grams: Long Distance Dependencies

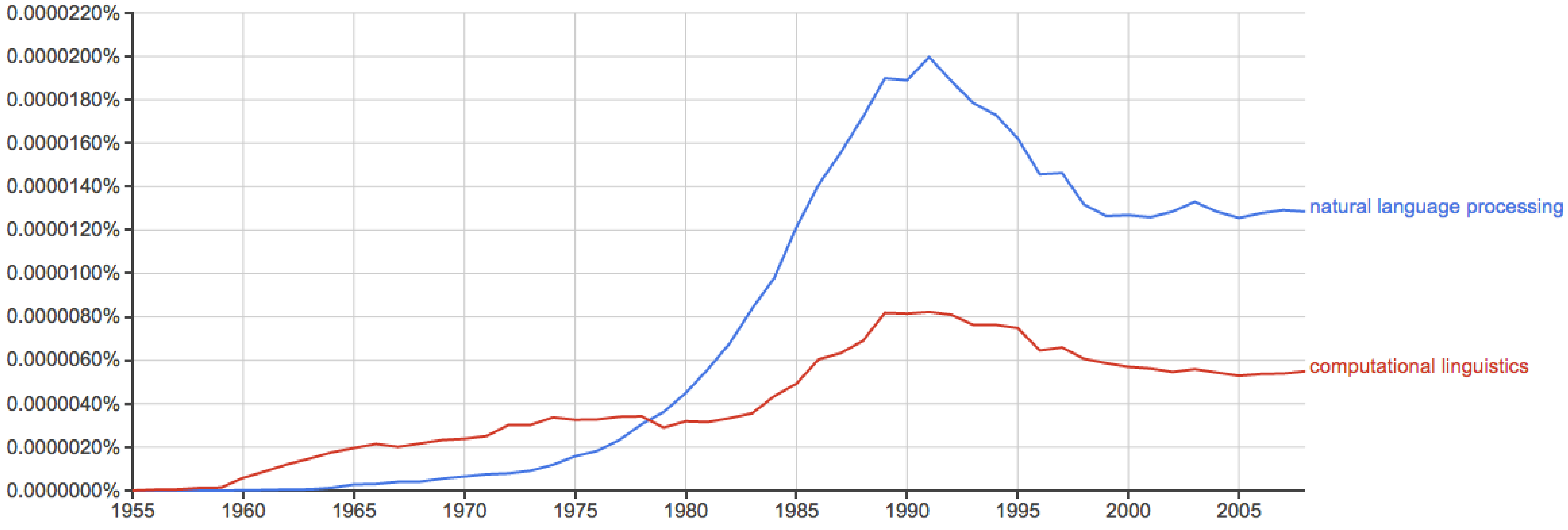
- Many times local context does not provide the most useful predictive clues, which instead are provided by ***long-distance dependencies***.
  - Syntactic dependencies
    - "The ***man*** next to the large oak tree near the grocery store on the corner **is** tall."
    - "The ***men*** next to the large oak tree near the grocery store on the corner **are** tall."
  - Semantic dependencies
    - "The ***bird*** next to the large oak tree near the grocery store on the corner **flies** rapidly."
    - "The ***man*** next to the large oak tree near the grocery store on the corner **talks** rapidly."
- More complex models of language are needed to handle such dependencies.


Google labs
Books Ngram Viewer
Graph these case-sensitive comma-separated phrases: phone,computer
between 1920 and 2008 from the corpus English with smoothing of 3.
Search lots of books
phone
computer
0.01000%
0.00900%
0.00800%
0.00700%
0.00600%
0.00500%
0.00400%
0.00300%
0.00200%
0.00100%
0.00000%
1920
1930
1940
1950
1960
1970
1980
1990
2000

Google Books Ngram Viewer

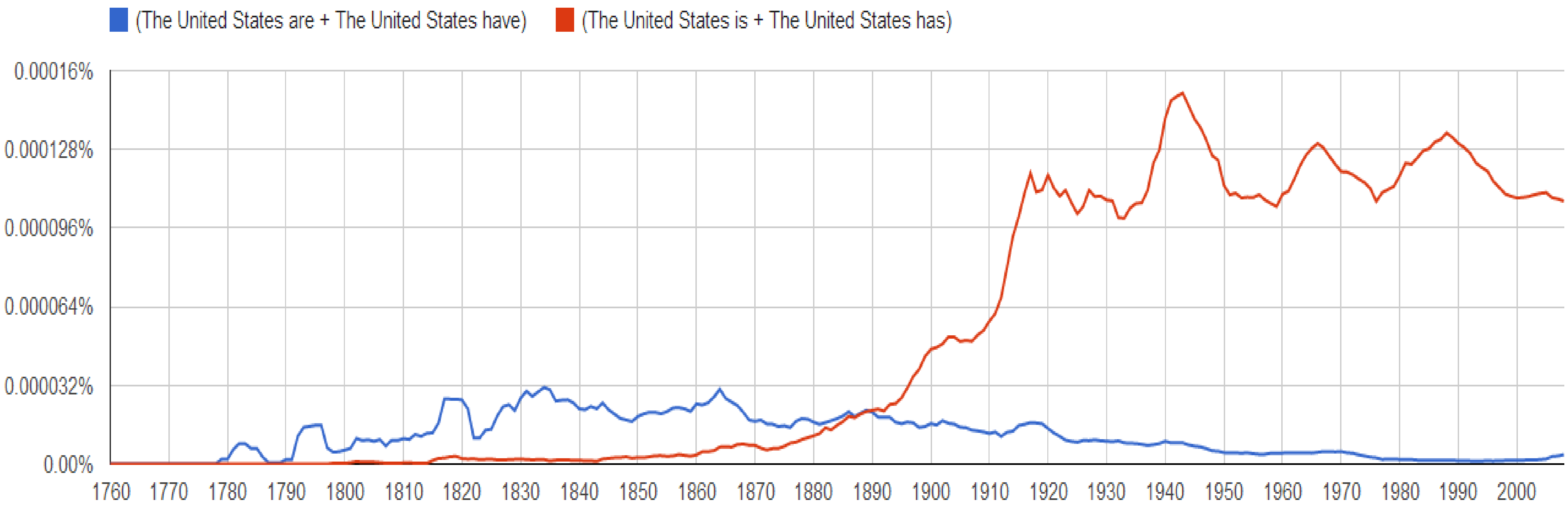
Graph these comma-separated phrases: natural language processing,computational linguistics case-insensitive

between 1955 and 2008 from the corpus English with smoothing of 3. Search lots of books



Frequency of Singular vs. Plural Usage of "United States" Over Time

Data collected using Google's N-gram Viewer



Thank you! Q&A

- SRILM:
  - www.speech.sri.com/projects/srilm
- Google N-Gram Release, August 2006, dataset details:
  - Over a trillion words
  - Over a billion 5-grams (c >= 40)
  - Over 13 million unique words (c >= 200)
- Google Books n-gram viewer:
  - http://ngrams.googlelabs.com

Questions?