CS 1671/2071
Human Language Technologies

Session 6: N-gram language models, part 1

Michael Miller Yoder

January 29, 2025

University of Pittsburgh | School of Computing and Information

Course logistics

- First quiz on Canvas due **tomorrow, Thu Jan 30**
    - Looking over the reading is a great way to prepare
- Project idea submission form is due **tomorrow, Thu Jan 30**
    - Check out the example projects on the project website
- I will release Homework 2 tomorrow or Friday

Overview: N-gram language models, part 1

- Language modeling
- N-gram language models
- Estimating n-gram probabilities
- Perplexity and evaluating language models
- Coding activity: build your own n-gram language model!

Structure of this course

MODULE 1	Prerequisite skills for NLP	text normalization, linear alg., prob., machine learning

	Approaches	How text is represented	NLP tasks
MODULE 2	statistical machine learning	n-grams	$\boxed{\text{language modeling}}$ text classification
MODULE 3			
MODULE 4			language modeling text classification sequence labeling

MODULE 5	NLP applications and ethics	machine translation, chatbots, information retrieval, bias

Introduction to language models

Language Models Estimate the Probability of Sequences

Which of these sentences would you be more likely to observe in an English corpus?

- Hugged I big brother my.
- I hugged my large brother.
- I hugged my big brother.

Slide credit: David Mortensen

Language Models Estimate the Probability of Sequences

Which of following word would be most likely to come after "David hates visiting New…"

- York
- California
- giggled

Slide credit: David Mortensen

These are actually instances of the same problem: the language modeling problem!

Slide credit: David Mortensen

Language Modeling is Tremendously Useful

LMs (language models) are at the center of NLP today and have many different applications

- Machine Translation
  P(high winds tonight) > P(large winds tonight)
- Spelling Correction
  P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Text Input Methods
  P(i cant believe how hot you **are**) > P(i cant believe how hot you **art**)
- Speech Recognition
  P(recognize speech) > P(wreck a nice beach)

Slide credit: David Mortensen

The Goal of Language Modeling

Compute the probability of a sequence of words/tokens/characters:

$$P(\boldsymbol{w}) = P(w_1, w_2, w_3, w_5, \ldots, w_n)$$

$$P(\text{I, hugged, my, big, brother})$$

This is related to next-word prediction:

$$P(w_t | w_1 w_2 \ldots w_{t-1})$$

$$P(\text{York}|\text{David, hates, going, to, New})$$

Do you compute either of these? Then you're in luck:

You are a language model!

Slide credit: David Mortensen

N-gram language models

The Chain Rule Helps Us Compute Joint Probabilities

The definition of conditional probability is

$$P(B|A) = \frac{P(A,B)}{P(A)}$$

which can be rewritten as

$$P(A,B) = P(A)P(B|A)$$

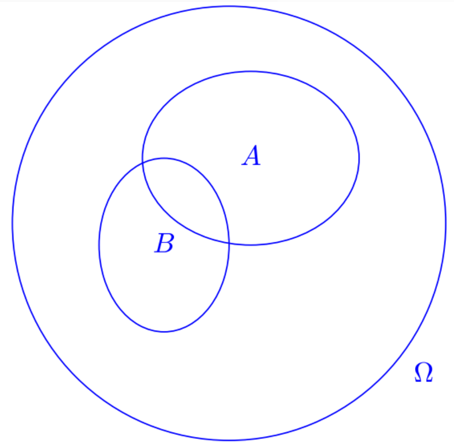

Figure 4.1: Events on the dart board

Slide credit: David Mortensen

The Chain Rule Helps Us Compute Joint Probabilities

If we add more variables, we see the following pattern:

$$
\begin{aligned}
P(A, B, C) &= P(A)P(B|A)P(C|A, B) \\
P(A, B, C, D) &= P(A)P(B|A)P(C|A, B)P(D|A, B, C)
\end{aligned}
$$

which can be generalized as

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\ldots P(x_n|x_1, \ldots, x_{n-1})$$

The Chain Rule!

Slide credit: David Mortensen

The chain rule to compute the joint probability of words in a sentence

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_i^n P(w_i | w_1 w_2 \ldots w_{i-1})$$

$$\begin{aligned} P(\text{now is the winter of our discontent}) = \\ P(\text{now}) \times P(\text{is}|\text{now}) \times \\ P(\text{the}|\text{now is}) \times P(\text{winter}|\text{now is the}) \times \\ P(\text{of}|\text{now is the winter}) \times \\ P(\text{our}|\text{now is the winter of}) \times \\ P(\text{discontent}|\text{now is the winter of our}) \end{aligned}$$

Slide credit: David Mortensen

Applying Chain Rule to LM

$$P(x_1, x_2, x_3, \dots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1, x_2, \dots x_{n-1})$$

$$P(w_1, w_2, w_3, \dots w_n) = \prod_i P(w_i|w_1, w_2 \dots w_{i-1})$$

How to compute P(its, water, is, so, transparent) ?

How Are We Estimating these Probabilities?

Could we just count and divide?

$$P(\text{discontent}|\text{now is the winter of our}) = \frac{\text{Count(now is the winter of our discontent)}}{\text{Count(now is the winter of our)}}$$

But this can't be a valid estimate! "now is the winter of our" is going to very rare in corpora. It isn't going to be a good estimate of its true probability.

Slide credit: David Mortensen

This May not Seem Very Helpful

Is $P(\text{discontent}|\text{now is the winter of our})$ really easier to compute than $P(\text{now is the winter of our discontent})$?

How can the chain rule help us? We can **cheat**.

Enter a Hero: Andrei Markov

Born	20 December 1978 (age 43) Voskresensk, Russian SFSR, Soviet Union
Height	6 ft 0 in (183 cm)
Weight	203 lb (92 kg; 14 st 7 lb)
Position	Defence
Played for	Khimik Voskresensk Dynamo Moscow Montreal Canadiens Vityaz Chekhov Ak Bars Kazan Lokomotiv Yaroslavl
Playing career	1995–2020

Or, Rather, Andrey Markov

Born	14 June 1856 N.S. Ryazan, Russian Empire
Died	20 July 1922 (aged 66) Petrograd, Russian SFSR
Known for	Markov chains; Markov processes; stochastic processes
Fields	Mathematics, specifically probability theory and statistics
Doctoral advisor	Pafnuty Chebyshev

Slide credit: David Mortensen

Markov Did a Computational Linguistics

Interestingly, Markov's first application of his idea of **Markov Chains** was to language, specifically to modeling alliteration and rhyme in Russian poetry.

As such, he can be seen not only as a great mathematician and statistician, but also one of the forerunners of **computational linguistics** and **computational humanities**.

Slide credit: David Mortensen

Markov Showed that You Could Make a Simplifying Assumption

One can approximate

$$P(\text{discontent}|\text{now is the winter of our})$$

by computing

$$P(\text{discontent}|\text{our})$$

or perhaps

$$P(\text{discontent}|\text{of our})$$

- We only get an estimate this way, but we can obtain it by only counting simpler things: “our discontent”, “discontent”, “of our”, etc
- N-gram language modeling is a generalization of this observation

Slide credit: David Mortensen

Markov Assumption


Andrei Markov

$$P(w_1, w_2, w_3, \dots w_n) = \prod_i P(w_i | w_1, w_2 \dots w_{i-1})$$

Simplify

$$P(w_1, w_2, w_3, \dots w_n) = \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

Slide credit:
Lorraine Li

This assumption is the Markov assumption

$$P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i | w_{i-k} w_{i-1})$$

In other words, we approximate each component in the product:

$$P(w_i | w_1, w_2, \ldots, w_{i-1}) \approx P(w_i | w_{i-k} \ldots w_{i-1})$$

We will now walk through what this looks like for different values of k.

Slide credit: David Mortensen

The Unigram Model ($k = 1$)

$$P(w_1 w_2 \dots w_i) \approx \prod_i P(w_i)$$

The probability of a sequence is approximately the product of the probabilities of the individual words.

Some automatically generated sequences from a unigram model:

- fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass
- thrift, did, eighty, said, hard, 'm, july, bullish
- that, or, limited, the

What do you notice about them?

The Bigram Model ($k = 2$)

If you condition on the previous word, you get the following:

$$P(w_i|w_1 w_2 \ldots w_{i-1}) \approx P(w_i|w_{i-1})$$

Some examples generated by a bigram model:

- texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen
- outside, new, car, parking, lot, of, the, agreement, reached
- this, would, be, a, record, november

Are these better?

The Trigram Model

The trigram model is just like the bigram model, only with a larger k:

$$P(w_i|w_1w_2\ldots w_{i-1}) \approx P(w_i|w_{i-2}w_{i-1})$$

The output of a trigram language model is generally **much** better than that of a bigram model **provided the training corpus is large enough**. Why do you need a larger corpus to train a trigram corpus than a bigram or unigram corpus?

N-gram models have trouble with long-range dependencies

In general, n-gram models are very impoverished models of language. For example, language has relationships that span many words:

- The **students** who worked on the assignment for three hours straight ***is/are** finally resting.
- The **teacher** who might have suddenly and abruptly met students **is/*are** tall.
- Violins are easy to mistakenly think you can learn to play ***them/quickly**.

Ngram LMs Are Often Adequate

Nevertheless, for many applications, ngram models are good enough (and they're super fast and efficient)

Estimating n-gram probabilities

Estimating bigram probabilities with the maximum likelihood estimate (MLE)

MLE for bigram probabilities can be computed as:

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

which we will sometimes represent as

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

Slide credit: David Mortensen

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>

$P(\texttt{I} \mid \texttt{<s>}) =$ $\quad$ $P(\texttt{Sam} \mid \texttt{<s>}) =$ $\quad$ $P(\texttt{am} \mid \texttt{I}) =$

$P(\texttt{</s>} \mid \texttt{Sam}) =$ $\quad$ $P(\texttt{Sam} \mid \texttt{am}) =$ $\quad$ $P(\texttt{do} \mid \texttt{I}) =$

Slide adapted from Jurafsky & Martin

More examples: Berkeley Restaurant Project sentences

can you tell me about any good cantonese restaurants close by

mid priced thai food is what i’m looking for

tell me about chez panisse

can you give me a listing of the kinds of food that are available

i’m looking for a good place to eat breakfast

when is caffe venezia open during the day

Slide adapted from Jurafsky & Martin

Raw bigram counts

Out of 9222 sentences

	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

Slide adapted from Jurafsky & Martin

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

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

Slide adapted from Jurafsky & Martin

Multiplication Considered Harmful

Doing computation in log space is preferred for language models

- **Avoid underflow** Multiplying small probabilities by small probabilities results in *very small* numbers, which is problematic
- **Optimize computation** Addition is cheaper than multiplication

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

Slide adapted from David Mortensen

The are high-performance toolkits for n-gram language modeling

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

Slide credit: David Mortensen

Perplexity and evaluating language models

The Evaluation Process for ML Models

The goal of LM evaluation:

- Does our model prefer good sentences to bad sentences?
- Specifically, does it assign higher probabilities to the good/grammatical/frequently observed ones and lower probabilities to the bad/ungrammatical/seldom observed ones?

In ML evaluation, we divide our data into three sets: `train`, `dev`, and `test`.

- We train the model's parameters on the `train` set
- We tune the model's hyperparameters (if appropriate) on the `dev` set (which should not overlap with the `train` set
- We test the model on the `test` set, which should not overlap with `train` or `dev`

An evaluation metric tells us how well our model has done on `test`.

We Can Evaluate Models Intrinsically or Extrinsically

- **Extrinsic Evaluation** means asking how much the model contributes to a larger task or goal. We may evaluate an LM based on how much it improves machine translation over a BASELINE.
- **Intrinsic Evaluation** means measuring some property of the model directly. We may quantify the probability that an LM assigns to a corpus of text.

In general, EXTRINSIC EVALUATION is better, but more expensive and time-consuming.

Slide credit: David Mortensen

Extrinsic Evaluation of LMs

Best evaluation for comparing models A and B

- Put each model in a task (spelling corrector, speech recognizer, MT system)
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly?
  - How many sentences translated correctly?
- Compare scores for A and B

This takes a lot of time to set up and can be expensive to carry out.

Perplexity is an intrinsic metric for language modeling

Perplexity evaluates the probability assigned by a model **to a collection of test documents, controlling for length** and is, thus, useful for evaluating LMs.

A better model of a text is one which assigns a higher probability to words that actually occur in the test set. This will result in **lower** perplexity.

However:

- It is a rather crude instrument
- It sometimes correlates only weakly with performance on downstream tasks
- It's only useful for pilot experiments
- But it's cheap and easy to compute, so it's important to understand

Slide credit: David Mortensen

Deriving Perplexity for Bigrams

$$PP(\mathbf{w}) = P(w_1 w_2 \dots w_n)^{-\frac{1}{n}}$$ Definition

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

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 w_2 \dots w_{i-1})}}$$ Chain Rule

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i)}}$$ For Unigrams

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_{i-1})}}$$ For Bigrams

To minimize perplexity is to maximize probability!

In general, a lower perplexity implies a better model.

Slide credit: David Mortensen

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Slide adapted from Jurafsky & Martin

Coding activity: build your own n-gram LM

N-gram language model on JupyterHub

- [Click on this nbgitpuller link]()
- Or find the link on the course website
- Open **session6_ngram_lm.ipynb**

CS 1671/2071 Human Language Technologies

School of Computing and Information, University of Pittsburgh
Spring 2025

Photo by Frits de Jong, art by Tjesje

Time	MW 1-2:15pm
Location	IS 405
Instructor	Michael Miller Yoder, PhD. *Please call me "Michael"*
Instructor contact	mmyoder@pitt.edu or through Canvas messages
Instructor office hours	By appointment in person at IS 604B or on Zoom Book an appointment
TA	Norah Almousa
TA contact	nia135@pitt.edu
TA office hours	By appointment
Textbook (free online)	[J+M] Jurafsky and Martin, *Speech and Language Processing*, 3e draft, 2024-08-20
Class notebooks	CRCD JupyterHub nbgitpuller link (Pitt login required, public link to source here).

Questions?