N-gram Language Modeling

Introduction to N-gram Language Models

Predicting words

The water of Walden Pond is beautifully ...

blue
green
clear

*refrigerator
*that

Language Models

Systems that can predict upcoming words

- Can assign a probability to each potential next word
- Can assign a probability to a whole sentence

Why word prediction?

It's a helpful part of language tasks

- Grammar or spell checking

Their are two midterms ~~Their~~ There are two midterms

Everything has improve Everything has ~~improve~~ improved

- Speech recognition

I will be back soonish I will be bassoon dish

Why word prediction?

It's how **large language models (LLMs)** work!

LLMs are **trained** to predict words

- Left-to-right (autoregressive) LMs learn to predict next word

LLMs **generate** text by predicting words

- By predicting the next word over and over again

Language Modeling (LM) more formally

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

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

Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4) \text{ or } P(w_n|w_1,w_2 \ldots w_{n-1})$$

An LM computes either of these:

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

How to estimate these probabilities

Could we just count and divide?

$$P(\texttt{blue} \mid \texttt{The water of Walden Pond is so beautifully}) = \frac{C(\texttt{The water of Walden Pond is so beautifully blue})}{C(\texttt{The water of Walden Pond is so beautifully})}$$

No! Too many possible sentences!

We'll never see enough data for estimating these

How to compute $P(W)$ or $P(w_n | w_1, \ldots w_{n-1})$

How to compute the joint probability $P(W)$:

P(The, water, of, Walden, Pond, is, so, beautifully, blue)

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

Reminder: The Chain Rule

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:

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

The Chain Rule in General

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

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

$$
\begin{aligned}
P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_{1:k-1})
\end{aligned}
$$

P("The water of Walden Pond") =

P(The) × P(water|The) × P(of|The water)

× P(Walden|The water of) × P(Pond|The water of Walden

Markov Assumption

Simplifying assumption:

Andrei Markov

$$P(\texttt{blue}|\texttt{The water of Walden Pond is so beautifully})$$

$$\approx \; P(\texttt{blue}|\texttt{beautifully})$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

Wikimedia commons

Bigram Markov Assumption

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

Instead of: $\prod_{k=1}^{n} P(w_k | w_{1:k-1})$

More generally, we approximate each component in the product

$$P(w_n | w_{1:n-1}) \approx P(w_n | w_{n-N+1:n-1})$$

Simplest case: Unigram model

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

Some automatically generated sentences from two different unigram models

```
To him swallowed confess hear both . Which . Of save on trail
for are ay device and rote life have

Hill he late speaks ; or ! a more to leg less first you enter

Months the my and issue of year foreign new exchange's September

were recession exchange new endorsed a acquire to six executives
```

Bigram model

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

Some automatically generated sentences rom two different unigram models

```
Why dost stand forth thy canopy, forsooth; he is this palpable hit
the King Henry. Live king. Follow.

What means, sir. I confess she? then all sorts, he is trim, captain.

Last December through the way to preserve the Hudson corporation N.
B. E. C. Taylor would seem to complete the major central planners
one gram 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
```

Problems with N-gram models

- N-grams can't handle **long-distance dependencies**:

"**The soups** that I made from that new cookbook I bought yesterday **were** amazingly delicious."

- N-grams don't do well at modeling new sequences with similar meanings

The solution: **Large language models**

- can handle much longer contexts
- because of using embedding spaces, can model synonymy better, and generate better novel strings

Why N-gram models?

A nice clear paradigm that lets us introduce many of the important issues for **large language models**

- **training** and **test** sets
- the **perplexity** metric
- **sampling** to generate sentences
- ideas like **interpolation** and **backoff**

N-gram Language Modeling

Introduction to N-grams

N-gram Language Modeling

Estimating N-gram Probabilities

Estimating bigram probabilities

The Maximum Likelihood Estimate

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

An example

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

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

More examples:
Berkeley Restaurant Project sentences

can you tell me about any good cantonese restaurants close by

tell me about chez panisse

i’m looking for a good place to eat breakfast

when is caffe venezia open during the day

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

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

$P(\text{<s> I want english food </s>}) =$

$P(\text{I}|\text{<s>})$

$\times\ P(\text{want}|\text{I})$

$\times\ P(\text{english}|\text{want})$

$\times\ P(\text{food}|\text{english})$

$\times\ P(\text{</s>}|\text{food})$

$=\ .000031$

What kinds of knowledge do N-grams represent?

$P(\text{english}|\text{want}) = .0011$

$P(\text{chinese}|\text{want}) = .0065$

$P(\text{to}|\text{want}) = .66$

$P(\text{eat} \mid \text{to}) = .28$

$P(\text{food} \mid \text{to}) = 0$

$P(\text{want} \mid \text{spend}) = 0$

$P(\text{i} \mid \text{<s>}) = .25$

Dealing with scale in large n-grams

LM probabilities are stored and computed in log format, i.e. **log probabilities**

This avoids underflow from multiplying many small numbers

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

If we need probabilities we can do one exp at the end

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

Larger ngrams

4-grams, 5-grams

Large datasets of large n-grams have been released

- N-grams from Corpus of Contemporary American English (COCA) 1 billion words (Davies 2020)
- Google Web 5-grams (Franz and Brants 2006) 1 trillion words)
- Efficiency: quantize probabilities to 4-8 bits instead of 8-byte float

Newest model: infini-grams (∞-grams) (Liu et al 2024)

- No precomputing! Instead, store 5 trillion words of web text in **suffix arrays**. Can compute n-gram probabilities with any n!

N-gram LM Toolkits

SRILM

- http://www.speech.sri.com/projects/srilm/

KenLM

- https://kheafield.com/code/kenlm/

N-gram Language Modeling

Estimating N-gram Probabilities

Language Modeling

Evaluation and Perplexity

How to evaluate N-gram models

"Extrinsic (in-vivo) Evaluation"

To compare models A and B

1. Put each model in a real task
   - Machine Translation, speech recognition, etc.
2. Run the task, get a score for A and for B
   - How many words translated correctly
   - How many words transcribed correctly
3. Compare accuracy for A and B

Intrinsic (in-vitro) evaluation

Extrinsic evaluation not always possible

- Expensive, time-consuming
- Doesn't always generalize to other applications

Intrinsic evaluation: **perplexity**

- Directly measures language model performance at predicting words.
- Doesn't necessarily correspond with real application performance
- But gives us a single general metric for language models
- Useful for large language models (LLMs) as well as n-grams

Training sets and test sets

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; different from training set.
  - Intuition: we want to measure generalization to unseen data
- An **evaluation metric** (like **perplexity**) tells us how well our model does on the test set.

Choosing training and test sets

- If we're building an LM for a specific task
  - The test set should reflect the task language we want to use the model for
- If we're building a general-purpose model
  - We'll need lots of different kinds of training data
  - We don't want the training set or the test set to be just from one domain or author or language.

Training on the test set

We can’t allow test sentences into the training set

- Or else the LM will assign that sentence an artificially high probability when we see it in the test set
- And hence assign the whole test set a falsely high probability.
- Making the LM look better than it really is

This is called “Training on the test set”

Bad science!

Dev sets

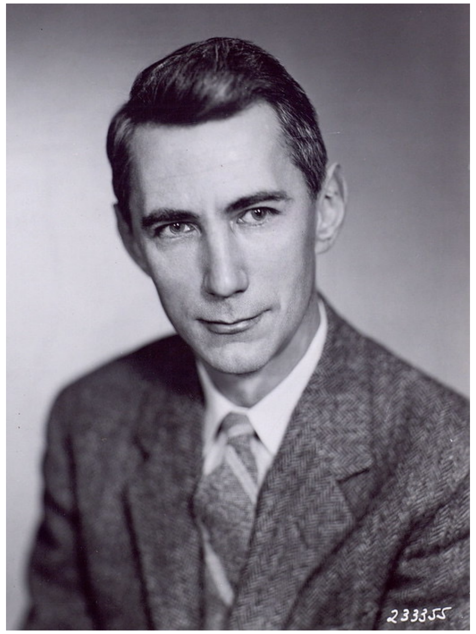
- If we test on the test set many times we might implicitly tune to its characteristics
  - Noticing which changes make the model better.
- So we run on the test set only once, or a few times
- That means we need a third dataset:
  - A **development test set** or, **devset**.
  - We test our LM on the devset until the very end
  - And then test our LM on the **test set** once

Intuition of perplexity as evaluation metric: How good is our language model?

Intuition: A good LM prefers "real" sentences

- Assign higher probability to “real” or “frequently observed” sentences
- Assigns lower probability to “word salad” or “rarely observed” sentences?

Intuition of perplexity 2: Predicting upcoming words



Claude Shannon

The Shannon Game: **How well can we predict the next word**?

- Once upon a ____
- That is a picture of a ____
- For breakfast I ate my usual ____

time	**0.9**
dream	**0.03**
midnight	**0.02**
…	
and	**1e-100**

Unigrams are terrible at this game (Why?)

A good LM is one that assigns a higher probability to the next word that actually occurs

Intuition of perplexity 3: The best language model is one that best predicts the entire unseen test set

- We said: a good LM is one that assigns a higher probability to the next word that actually occurs.
- Let's generalize to all the words!
  - The best LM assigns high probability to the entire test set.
- When comparing two LMs, A and B
  - We compute P_A(test set) and P_B(test set)
  - The better LM will give a higher probability to (=be less surprised by) the test set than the other LM.

Intuition of perplexity 4: Use perplexity instead of raw probability

- Probability depends on size of test set
  - Probability gets smaller the longer the text
  - Better: a metric that is **per-word**, normalized by length
- **Perplexity** is the inverse probability of the test set, normalized by the number of words

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

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

Intuition of perplexity 5: the **inverse**

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

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

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

(The inverse comes from the original definition of perplexity from cross-entropy rate in information theory)

Probability range is [0,1], perplexity range is [1,∞]

Minimizing perplexity is the same as maximizing probability

Intuition of perplexity 6: N-grams

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

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

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

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

Intuition of perplexity 7: Weighted average branching factor

Perplexity is also the **weighted average branching factor** of a language.

Branching factor: number of possible next words that can follow any word

Example: Deterministic language L = {red,blue, green}

Branching factor = 3 (any word can be followed by red, blue, green)

Now assume LM A where each word follows any other word with equal probability ⅓

Given a test set T = "red red red red blue"

$\text{Perplexity}_A(T) = P_A(\text{red red red red blue})^{-1/5} = \quad ((⅓)^5)^{-1/5} \quad = (⅓)^{-1} \quad =3$

But now suppose red was very likely in training set, such that for LM B:

- P(red) = .8   p(green) = .1 p(blue) = .1

We would expect the probability to be higher, and hence the perplexity to be smaller:

$\text{Perplexity}_B(T) = P_B(\text{red red red red blue})^{-1/5}$

$= (.8 * .8 * .8 * .8 * .1)^{-1/5} \quad =.04096^{-1/5} \quad = .527^{-1} \quad = 1.89$

Holding test set constant:
Lower perplexity = better language model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

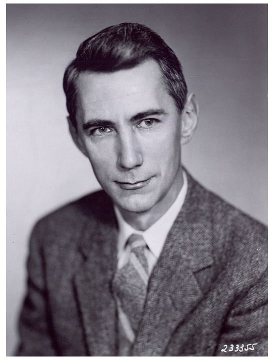
Language Modeling

Evaluation and Perplexity

Sampling and Generalization

Language Modeling

The Shannon (1948) Visualization Method
Sample words from an LM



Claude Shannon

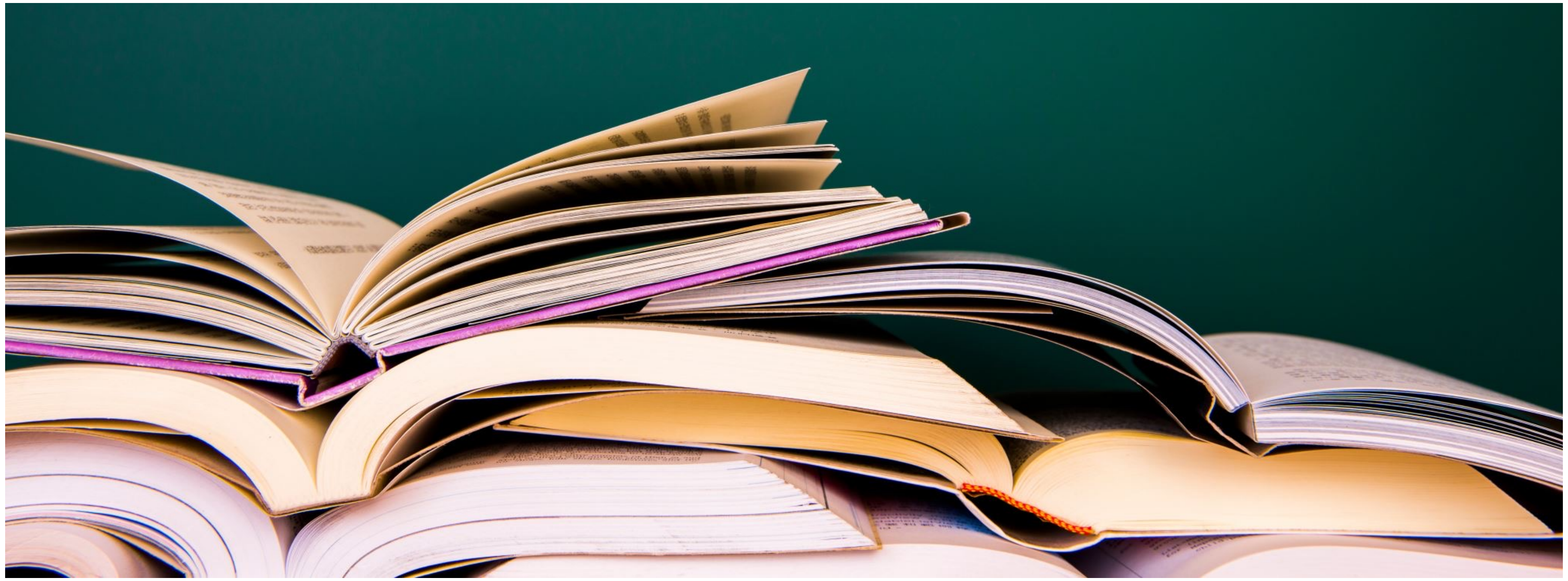
Unigram:

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

Bigram:

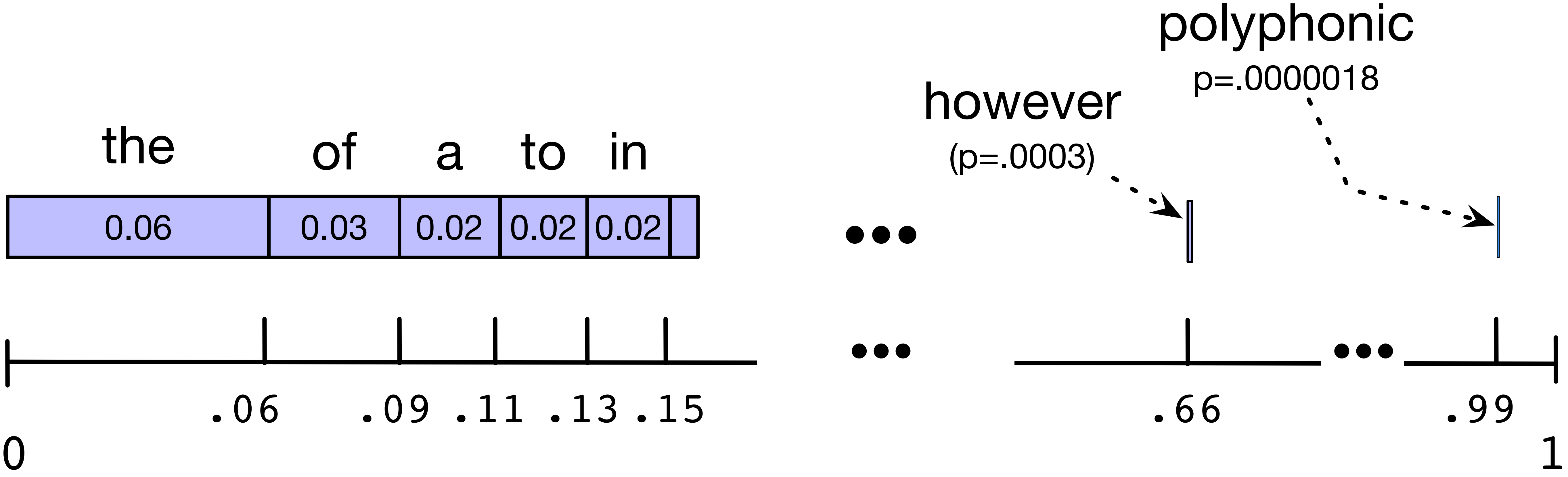
THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

How Shannon sampled those words in 1948



"Open a book at random and select a letter at random on the page. This letter is recorded. The book is then opened to another page and one reads until this letter is encountered. The succeeding letter is then recorded. Turning to another page this second letter is searched for and the succeeding letter recorded, etc."

Sampling a word from a distribution



Visualizing Bigrams the Shannon Way

Choose a random bigram (<s>, w)

according to its probability p(w|<s>)

Now choose a random bigram (w, x) according to its probability p(x|w)

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
```

Note: there are other sampling methods

Used for neural language models

Many of them avoid generating words from the very unlikely tail of the distribution

We'll discuss when we get to neural LM decoding:

- Temperature sampling
- Top-k sampling
- Top-p sampling

Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

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)
- That sparsity is even worse for 4-grams, explaining why our sampling generated actual Shakespeare.

The Wall Street Journal is not Shakespeare

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

Can you guess the author? These 3-gram sentences are sampled from an LM trained on who?

1) 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 gram Brazil on market conditions

2) This shall forbid it should be branded, if renown made it empty.

3) "You are uniformly charming!" cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Choosing training data

If task-specific, use a training corpus that has a similar genre to your task.

- If legal or medical, need lots of special-purpose documents

Make sure to cover different kinds of dialects and speaker/authors.

- Example: *African-American Vernacular English (AAVE)*
  - One of many varieties that can be used by African Americans and others
  - Can include the auxiliary verb **finna** that marks immediate future tense:
  - "My phone finna die"

The perils of overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus

- But even when we try to pick a good training corpus, the test set will surprise us!
- 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

Zeros

Training set:

… ate lunch
… ate dinner
… ate a
… ate the

P("breakfast" | ate) = 0

- Test set

  … ate lunch
  … ate breakfast

Zero probability bigrams

Bigrams with zero probability

- Will hurt our performance for texts where those words appear!
- And mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!

Language Modeling

Sampling and Generalization

N-gram Language Modeling

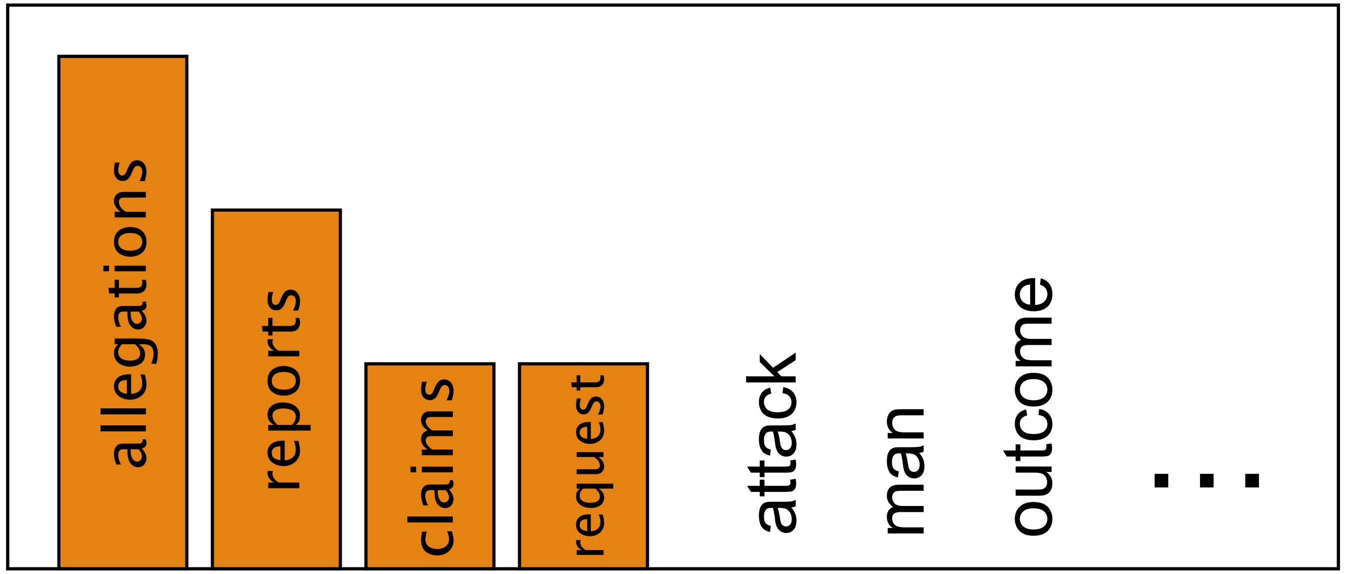
Smoothing, Interpolation, and Backoff

The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

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

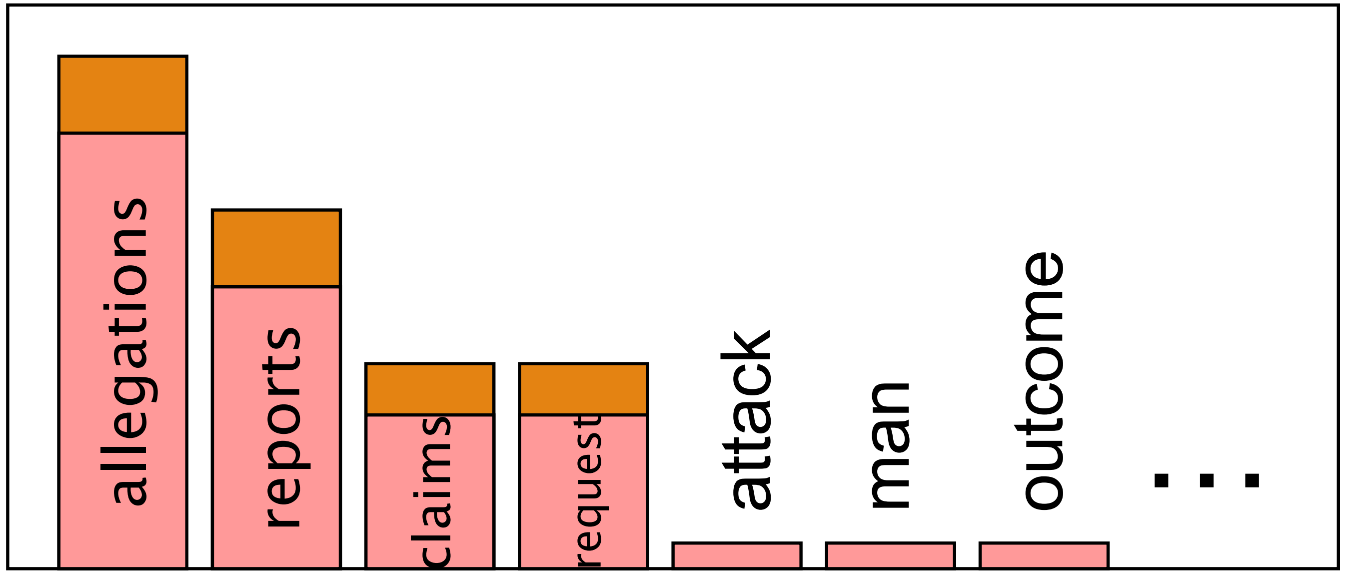
7 total



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



Add-one estimation

Also called Laplace smoothing

Pretend we saw each word one more time than we did

Just add one to all the counts!

MLE estimate:

$$P_{\text{MLE}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Add-1 estimate:

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

Maximum Likelihood Estimates

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word “bagel” occurs 400 times in a corpus of a million words

What is the probability that a random word from some other text will be “bagel”?

MLE estimate is 400/1,000,000 = .0004

This may be a bad estimate for some other corpus

- But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	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

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	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

Reconstituted 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	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

Compare with raw bigram counts

	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

	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

Add-1 estimation is a blunt instrument

So add-1 isn't used for N-grams:

- Generally we use interpolation or backoff instead

But add-1 is used to smooth other NLP models

- For text classification
- In domains where the number of zeros isn't so huge.

Backoff and Interpolation

Sometimes it helps to use **less** context

- Condition on less context for contexts you know less 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

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

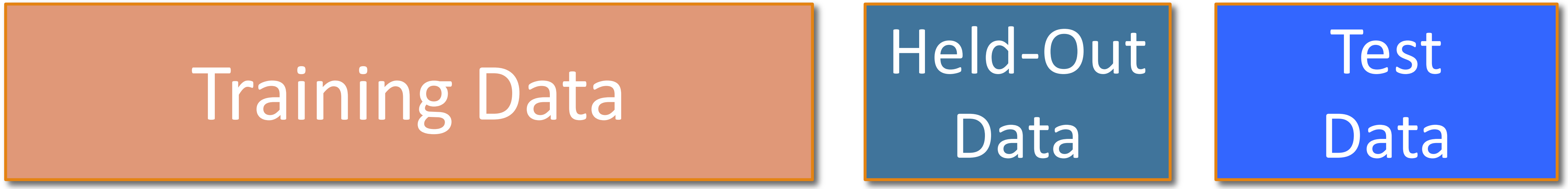
$$\sum_i \lambda_i = 1$$

Lambdas conditional on context:

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

How to set λs for interpolation?

Use a **held-out** corpus



Choose λs to maximize 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

Backoff

Suppose you want:

P(pancakes| delicious soufflé)

If the trigram probability is 0, use the bigram

P(pancakes| soufflé)

If the bigram probability is 0, use the unigram

P(pancakes)

Complication: need to discount the higher-order ngram so probabilities don't sum higher than 1 (e.g., Katz backoff)

Stupid Backoff

Backoff without discounting (not a true probability)

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

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

N-gram Language Modeling

Interpolation and Backoff