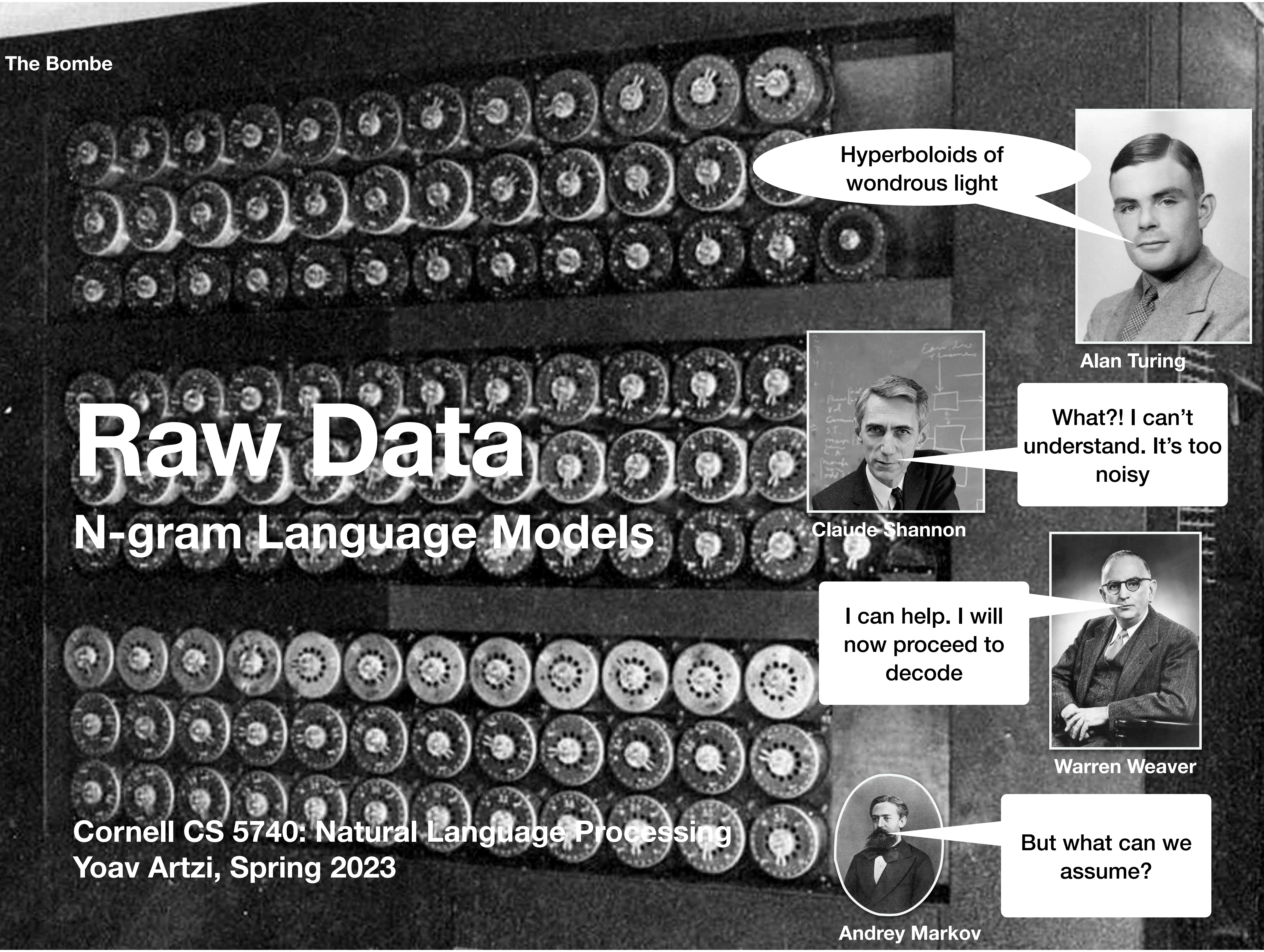


Raw Data

N-gram Language Models

Cornell CS 5740: Natural Language Processing
Yoav Artzi, Spring 2023

Language Models

From the Imitation Game (2014)

The Language Model Problem

- Let a vocabulary $\mathscr{V}$ be a finite set of tokens

$$\mathscr{V} = \{\text{the, a, man, telescope, Madrid, two, } \ldots\}$$

- We can construct an infinite set of sentences (i.e., strings) $\bar{x}$
- $\mathscr{V}^\dagger = \{\text{the, a, the a, the fan, the man, the man with the telescope, } \ldots\}$
- Given: a dataset of example sentence $\mathscr{D} = \{\bar{x}^{(i)}\}_{i=1}^{M}$
- Goal: estimate a probability distribution over sentences, s.t., $\sum_{\bar{x} \in \mathscr{V}^\dagger} p(\bar{x}) = 1$ and $p(\bar{x}) \geq 0$ for all $\bar{x} \in \mathscr{V}^\dagger$
- Question: why would we ever want to do this?

$p(\text{the}) = 10^{-12}$

$p(\text{a}) = 10^{-13}$

$p(\text{the fan}) = 10^{-12}$

$p(\text{the fan saw Beckham}) = 2 \times 10^{-8}$

$p(\text{the fan saw saw}) = 10^{-15}$

$\ldots$

Language Models Use

The Noisy Channel Model

- Goal: predict a sentence given some input $p(\bar{x} \mid a)$

- The noisy channel approach:

Language model: Distributions over sequences of words (sentences)

$$\bar{x}^* = \arg\max_{\bar{x} \in \mathcal{V}^\dagger} p(\bar{x} \mid a)$$

Input signal of some sorts (e.g., audio)

$$= \arg\max_{\bar{x} \in \mathcal{V}^\dagger} p(a \mid \bar{x}) p(\bar{x}) / p(a)$$

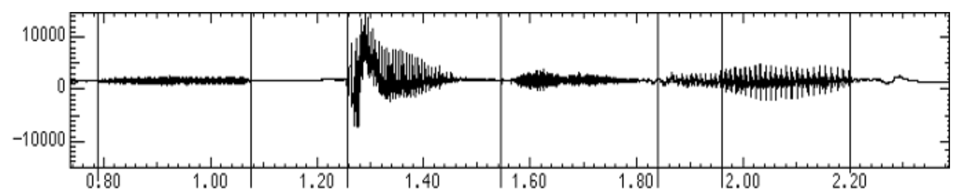
$$= \arg\max_{\bar{x} \in \mathcal{V}^\dagger} p(a \mid \bar{x}) p(\bar{x})$$

- So, if $p(a \mid \bar{x})$ is not great (i.e., noisy), $p(\bar{x})$ can compensate

The Noisy Channel Model

Speech Recognition

- Automatic speech recognition (ASR)
- Audio in a, text out $\bar{x}$



- "Wreck a nice beach?"
  - "Recognize speech"
- "Eye eight uh Jerry?"
  - "I ate a cherry"

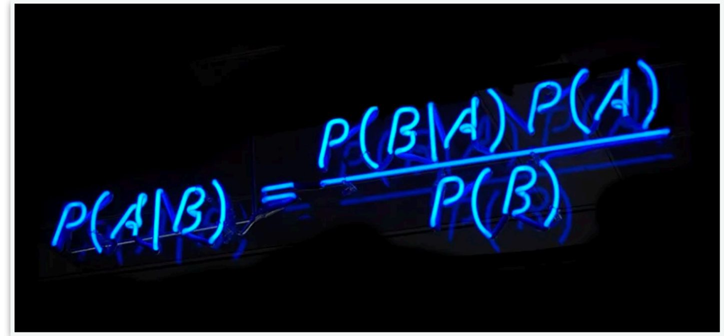
Speech Recognition

Acoustically Scored Hypotheses

the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

Speech Recognition

ASR Noisy Channel System



- Let a be an audio signal, $\bar{x}$ a sentence, and:
  - **Source** be a language model $p(\bar{x})$
  - **Channel** be an acoustic model $p(a \mid \bar{x})$
- We decode $\bar{x}$ from a using Bayes rule:

$$\arg\max_{\bar{x}} p(\bar{x} \mid a) = \arg\max_{\bar{x}} p(a \mid \bar{x}) p(\bar{x})$$

The Noisy Channel Model

Translation

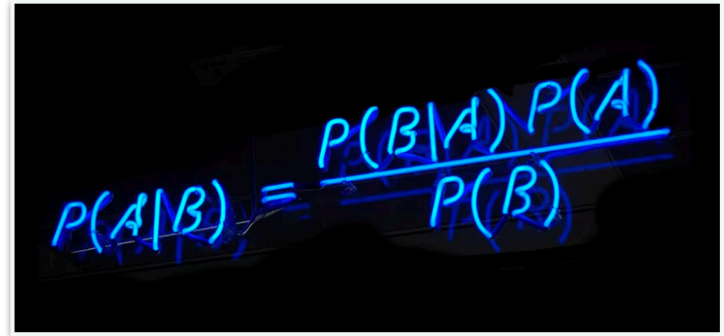
“Also knowing nothing official about, but having guessed and inferred considerable about, the powerful new mechanized methods in cryptography—methods which I believe succeed even when one does not know what language has been coded—one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: ‘This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.’ ”

Warren Weaver

(1955:18, quoting a letter he wrote in 1947)

Translation

MT Noisy Channel System



- Let f be a sentence in the source language, $\bar{x}$ a sentence in the target language, and:
  - **Source** be a language model $p(\bar{x})$
  - **Channel** be a translation model $p(f \mid \bar{x})$
- We decode $\bar{x}$ from f using Bayes rule:

$$\arg\max_{\bar{x}} p(\bar{x} \mid f) = \arg\max_{\bar{x}} p(f \mid \bar{x}) p(\bar{x})$$

Caption Generation

Captioning Noisy Channel System

- Let I be a sentence in the source language, $\bar{x}$ a sentence in the target language, and:
  - **Source** be a language model $p(\bar{x})$
  - **Channel** be an image model $p(I \mid \bar{x})$
- We decode $\bar{x}$ from I using Bayes rule:

$$\arg\max_{\bar{x}} p(\bar{x} \mid I) = \arg\max_{\bar{x}} p(I \mid \bar{x}) p(\bar{x})$$

Language Models Use

Universal Text Models

- Assume that any problem can be described as text-to-text:
  - What is the french translation of “I love Lucy”? → J'aime lucy
  - What is the sentiment of “I Love Lucy”? → Very positive
- Then a language model can conceptually solve it by just generating the answer as continuation
- So, language models can be universal text models
- Of course, that would have to be <u>a really good language model</u>

Language Models Use

Universal Text Models

FEBRUARY 14, 2019

Better Language Models and Their Implications

We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization — all without task-specific training.

VIEW CODE

READ PAPER

READ MORE

Learning Language Models

- Goal: estimate $p(\bar{x})$, where $\bar{x}$ is a natural language sentence
- Learning input: M observations of raw sentences $\bar{x}$
- Learning output: model to compute $p(\bar{x})$ for any $\bar{x}$
- Probabilities should broadly indicate sentence plausibility
  - $p($I saw a van$) >> p($eyes aw of an$)$
  - Not only grammaticality: $p($artichokes intimidate zippers$) \approx 0$
  - Generally, plausibility depends on context

Learning Language Models

- Goal: estimate $p(\bar{x})$, where $\bar{x}$ is a natural language sentence
- Learning input: M observations of raw sentences $\bar{x}$
- Learning output: model to compute $p(\bar{x})$ for any $\bar{x}$
- Option 1: empirical distribution over training sentences

$$p(\bar{x}) = \frac{c(\bar{x})}{M}, \text{ where } c \text{ is the counting function}$$

Learning Language Models

- Goal: estimate $p(\bar{x})$, where $\bar{x}$ is a natural language sentence
- Learning input: M observations of raw sentences $\bar{x}$
- Learning output: model to compute $p(\bar{x})$ for any $\bar{x}$
- Option 1: empirical distribution over training sentences

$$p(\bar{x}) = \frac{c(\bar{x})}{M}, \text{ where } c \text{ is the counting function}$$

- Problem: does not generalize at all!
  - Need to be able to assign non-zero probabilities to unseen sentences

Probability Decomposition

- Assume: the choice of each word x_i in $\bar{x} = \langle x_1, \ldots, x_n \rangle$ depends on previous words only

$$p(\bar{x}) = \prod_{i=1}^{n} p(x_i \,|\, x_1, \ldots, x_{i-1})$$

- Better?

Probability Decomposition

- Decompose using the **chain rule**: the choice of each word x_i in $\bar{x} = \langle x_1, \ldots, x_n \rangle$ depends on previous words only

$$p(\bar{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

- Better?
  - Yes, but not really: last word still represents the complete sentence event, and will zero everything
  - So, back to square one

Probability Decomposition

- Decompose using the **chain rule**: the choice of each word x_i in $\bar{x} = \langle x_1, \ldots, x_n \rangle$ depends on previous words only

$$p(\bar{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

- Better?

Probability Decomposition

Markov Assumption

- **Markov property** refers to the memoryless property of a stochastic process (i.e., future decision are independent of the past)

- A stochastic model can assume the Markov property

$$
\begin{aligned}
&p(\text{english} \mid \text{this is really in}) \approx \\
&p(\text{english} \mid \text{is really in}) \approx \\
&p(\text{english} \mid \text{really in}) \approx \\
&p(\text{english} \mid \text{in}) \approx \\
&p(\text{english})
\end{aligned}
$$

- It's a simplifying approximation — no free lunch!

Unigram Models

- The most crude approximation: unigrams

$$p(\bar{x}) = p(\langle x_1, \ldots, x_n \rangle) = \prod_{i=1}^{n} p(x_i)$$

where $x_i \in \mathcal{V} \cup \{\text{STOP}\}$

- Can easily compute the probability of a given sentence
- And can also generate!

$$\boxed{\begin{array}{l} i = 0 \\ \text{repeat} \\ \quad i++ \\ \quad x_i \sim p(x) \\ \text{until } x_i = \text{STOP} \\ \text{return } \langle x_1, \ldots, x_i \rangle \end{array}}$$

Unigram Models

- The most crude approximation: unigrams

$$p(\bar{x}) = p(\langle x_1, \ldots, x_n \rangle) = \prod_{i=1}^{n} p(x_i)$$

- Let's generate:
  - [thrift, did, eighty, said, hard, 'm, july, bullish]
  - []
  - [after, any, on, consistently, hospital, lake, of, of, other, and, factors, raised, analyst, too, allowed, mexico, never, consider, fall, bungled, davison, that, obtain, price, lines, the, to, sass, the, the, further, board, a, details, machinists, between, nasdaq]
- Why is it bad?

Unigram Models

- The most crude approximation: unigrams

$$p(\bar{x}) = p(\langle x_1, \ldots, x_n \rangle) = \prod_{i=1}^{n} p(x_i)$$

- Let's generate:
  - [thrift, did, eighty, said, hard, 'm, july, bullish]
  - []
  - [after, any, on, consistently, hospital, lake, of, of, other, and, factors, raised, analyst, too, allowed, mexico, never, consider, fall, bungled, davison, that, obtain, price, lines, the, to, sass, the, the, further, board, a, details, machinists, between, nasdaq]
- Why is it bad?
- Big problem with unigrams: $p(\text{the the the the}) >> p(\text{I like icecream})$

Bi-gram Models

- Relaxing the strict Markov assumption a bit: bi-grams

$$p(\bar{x}) = p(\langle x_1, \ldots, x_n \rangle) = \prod_{i=1}^{n} p(x_i \,|\, x_{i-1}), \text{ where } x_0 = *\, , x_i \in \mathscr{V} \cup \{\mathrm{STOP}\}$$

- Why do we need $x_0 = *$?

- Examples:

  - [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]

  - [although, common, shares, rose, forty, six, point, four, hundred, dollars, from, thirty, seconds, at, the, greatest, play, disingenuous, to, be, reset, annually, the, buy, out, of, american, brands, vying, for, mr., womack, currently, sharedata, incorporated, believe, chemical, prices, undoubtedly, will, be, as, much, is, scheduled, to, conscientious, teaching]

  - [this, would, be, a, record, november]

- No free lunch: what's the cost compared to unigram models?

N-gram Models

- N-gram models ($N > 1$) condition on $N-1$ previous words

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \,|\, x_{i-(N-1)}, \ldots, x_{i-1})$$

where $x_i \in \mathscr{V} \cup \{\mathrm{STOP}\}$ and $x_{-N+2}, \ldots, x_0 = *$

- Example 3-gram model:

$$p(\text{the dog barks STOP}) = p(\text{the} \,|\, *, *) \times p(\text{dog} \,|\, *, \text{the}) \times$$
$$p(\text{barks} \,|\, \text{the, dog}) \times p(\mathrm{STOP} \,|\, \text{dog, barks})$$

N-gram Models

Well-defined Distributions

- Simplest case: unigrams $p(\bar{x}) = p(\langle x_1, \ldots, x_n \rangle) = \prod_{i=1}^{n} p(x_i)$
- For all strings $\bar{x}$ (of any length): $p(\bar{x}) \geq 0$
- Need to show the sum over string of all lengths $\sum_{\bar{x}} p(\bar{x}) = 1$

(1) $$\sum_{\bar{x}} p(\bar{x}) = \sum_{n=1}^{\infty} \sum_{x_1, \ldots, x_n} p(x_1, \ldots, x_n)$$

(2) $$\sum_{x_1, \ldots, x_n} p(x_1, \ldots, x_n) = \sum_{x_1, \ldots, x_n} \prod_{i=1}^{n} p(x_i) = \sum_{x_1} \cdots \sum_{x_n} p(x_1) \times \ldots \times p(x_n)$$

$$= \sum_{x_1} p(x_1) \times \cdots \times \sum_{x_n} p(x_n) = (1 - p_s)^{n-1} \text{ where } p_s = p(\text{STOP})$$

(1)+(2) $$\sum_{\bar{x}} p(\bar{x}) = \sum_{n=1}^{\infty} (1 - p_s)^{n-1} p_s = p_s \sum_{n=1}^{\infty} (1 - p_s)^{n-1} = p_s \frac{1}{1 - (1 - p_s)} = 1$$

N-gram Models

Well-defined Distributions

Surprisingly neural network LMs are not necessarily well-define distributions (Chen et al. 2018)

- Simplest case: unigrams $p(\bar{x}) = p(\langle x_1, \ldots, x_n \rangle) = \prod_{i=1}^{n} p(x_i)$
- For all strings $\bar{x}$ (of any length): $p(\bar{x}) \geq 0$
- Need to show the sum over string of all lengths $\sum_{\bar{x}} p(\bar{x}) = 1$

(1) $$\sum_{\bar{x}} p(\bar{x}) = \sum_{n=1}^{\infty} \sum_{x_1,\ldots,x_n} p(x_1, \ldots, x_n)$$

(2) $$\sum_{x_1,\ldots,x_n} p(x_1, \ldots, x_n) = \sum_{x_1,\ldots,x_n} \prod_{i=1}^{n} p(x_i) = \sum_{x_1} \cdots \sum_{x_n} p(x_1) \times \ldots \times p(x_n)$$

$$= \sum_{x_1} p(x_1) \times \cdots \times \sum_{x_n} p(x_n) = (1 - p_s)^{n-1} \text{ where } p_s = p(\text{STOP})$$

(1)+(2) $$\sum_{\bar{x}} p(\bar{x}) = \sum_{n=1}^{\infty} (1 - p_s)^{n-1} p_s = p_s \sum_{n=1}^{\infty} (1 - p_s)^{n-1} = p_s \frac{1}{1 - (1 - p_s)} = 1$$

N-gram Models

Sampling from N-gram models

- N-gram models ($N > 1$) condition on $N - 1$ previous words

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \,|\, x_{i-(N-1)}, \ldots, x_{i-1})$$

where $x_i \in \mathscr{V} \cup \{\text{STOP}\}$ and $x_{-N+2}, \ldots, x_0 = *$

- Sampling generalizes easily from unigrams and up:

$$\boxed{\begin{array}{l} i = 0 \\ \texttt{repeat} \\ \quad i++ \\ \quad x_i \sim p(x_i \,|\, x_{i-(N-1)}, \ldots, x_{i-1}) \\ \texttt{until}\ x_i = \text{STOP} \\ \texttt{return}\ \langle x_1, \ldots, x_i \rangle \end{array}}$$

N-gram Models

Learning

- The parameters of N-gram models are the probabilities
- Maximum likelihood estimate has a closed-form solution: relative frequencies
- $q_{ML}(w) = \dfrac{c(w)}{c()}, \quad q_{ML}(w \mid v) = \dfrac{c(v,w)}{c(v)}, \quad q_{ML}(w \mid u,v) = \dfrac{c(u,v,w)}{c(u,v)}, \quad \ldots$
- where $c(), c(w), c(w,v), \ldots$ the empirical counts on the training set
- The general approach:
  - Take a training set D and test set D'
  - Compute the ML estimates using D
  - Use it to assign probabilities to other sentences, such as those in D'

$$p_{ML}(\text{door} \mid \text{the}) = \frac{14{,}112{,}454}{2{,}313{,}581{,}162} = 0.0006$$

Training Counts

198015222 the first

194623024 the same

168504105 the following

158562063 the world

…

14112454 the door

23135851162 the *

- Probabilities will be very small, so everything is done in log-space

N-gram Models

Learning

- As we increase N (higher-order N-grams), sparsity increases
- Counts becomes smaller and smaller, and there are more zeros

198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door

23135851162 the *

197302 close the window
191125 close the door
152500 close the gap
116451 close the thread
87298 close the deal

3785230 close the *

3380 please close the door
1601 please close the window
1164 please close the new
1159 please close the gate
…
0 please close the first

13951 please close the *

Please close the door

Please close the first window on the left

N-gram Models

Approximating Shakespeare

1-gram

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

N-gram Models

Shakespeare as a Corpus

- 884,647 tokens, vocabulary size of $|\mathscr{V}|$=29,066
- Shakespeare produced 300,000 bigram types out of $|\mathscr{V}|^2$= 844M possible bigrams
  - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Even worse with 4-grams: what's coming out looks like Shakespeare because it is Shakespeare
- More about this real soon … but first: evaluation

Evaluation

How Good is our Model?

- How good is our model? At what?
- The goal is not to just generate fake sentences!
  - That would be easy to do well
  - Higher order n-grams will always give better looking sentences
  - But they are just overfitting — why?
- We want our model to prefer **good** sentences over **bad** ones
  - Higher probability to real or frequent sentences
    - Than ungrammatical or rare ones
  - How does this relate to how we use the language model? For example, in a noisy channel transcription system

Evaluation

Testing

- We must test the model on data it hasn't seen during learning
  - Otherwise — overfitting! 😱
- We need an evaluation metric — two options:
  - Extrinsic: focused on however the model will be used — for example, can it improve a transcription system?
  - Intrinsic: focused on the language model task — how good can the model assign probabilities to real unseen data?
- Ideally, the two correlate, but reality is more complex

Extrinsic Evaluation

Word Error Rate (WER)

- Common metric for automatic speech transcription (ASR) evaluation
- Given an output $\bar{x}^*$ and a gold label $\bar{x}^{(i)}$:

$$\mathrm{WER}(\bar{\mathrm{x}}^*, \mathrm{x}^{(\mathrm{i})}) = \frac{\#\text{ insertions} + \#\text{ deletions} + \#\text{ substitutions}}{\#\text{ words in } \bar{\mathrm{x}}^{(\mathrm{i})}}$$

- Extrinsic measures are more credible, but limited to a specific use and are harder to deploy
  - You need the complete system, and often evaluating it is hard

$\bar{x}^{(i)}$: Andy saw a part of the movie

$\bar{x}^*$: And he saw apart of the movie

$$\mathrm{WER}(\bar{\mathrm{x}}^*, \mathrm{x}^{(\mathrm{i})}) = \frac{1+1+2}{7} = \frac{4}{7} = 57\ \%$$

Intrinsic Evaluation

The Shannon Game

- How well can we predict the next word?

  When I eat pizza, I wipe of the ____

  Many children are allergic to ____

  I saw a ____

grease	0.5
sauce	0.4
dust	0.05
…	
mice	0.0001
…	
the	1E-100

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

Evaluation

Perplexity

- The best language model is the one the is best at predicting the test set → will give test sentences the highest probability

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

- Given a set of test sentences D' with a total of m words:

$$PP(D') = p(D')^{-1/m} = (\prod_{\bar{x} \in D'} p(\bar{x}))^{-1/m}$$

- In practice, we work in log space:

$$PP(D') = 2^{-\frac{1}{m}\sum_{\bar{x} \in D'} \log_2 p(\bar{x})}$$

- Lower perplexity is better

- What happens if we give a test sentence zero probability? 🤯

Evaluation

Perplexity of a Uniform Model

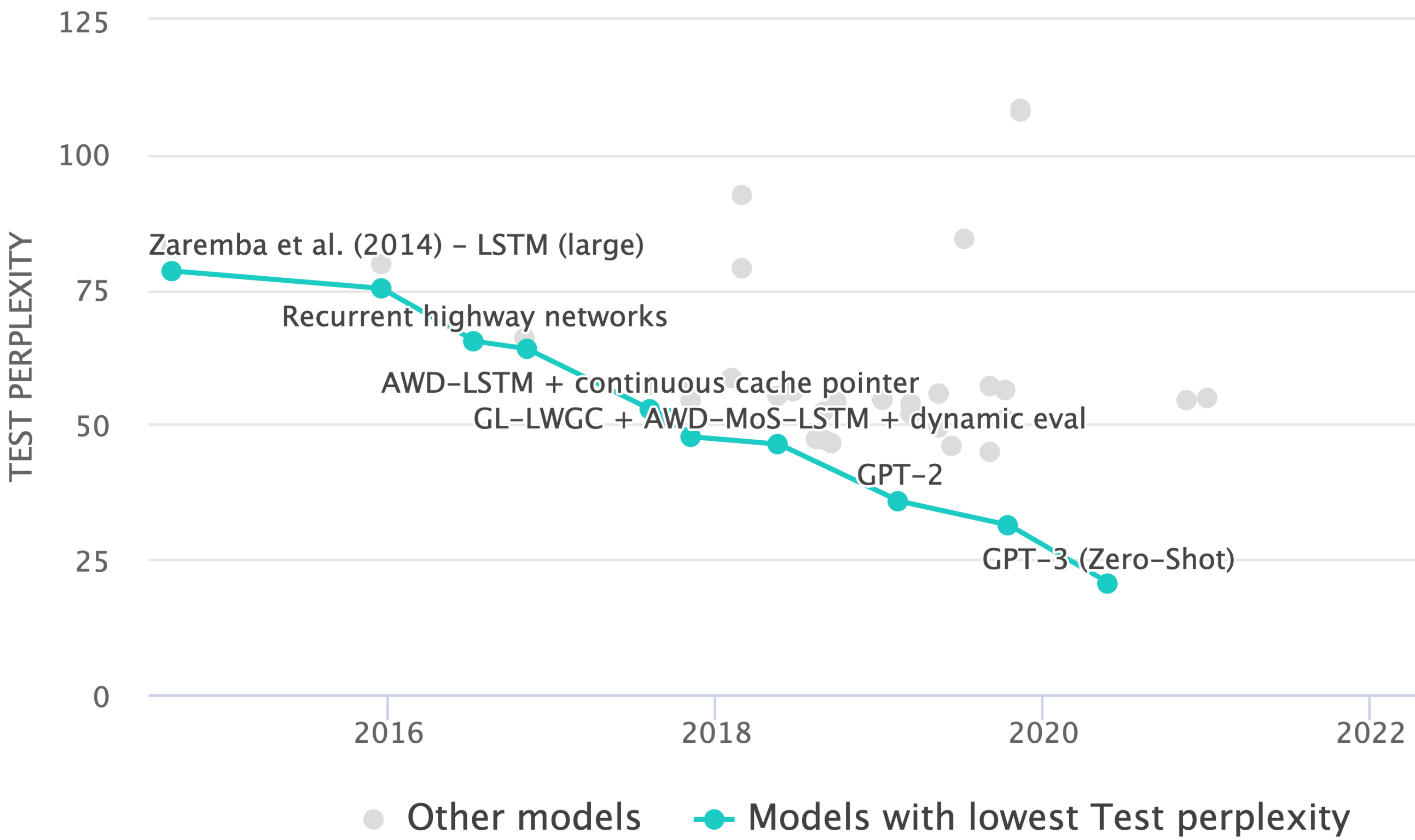
- Under a uniform distribution perplexity will be the vocabulary size
- Assume M sentences consisting of m random digits, $|\mathcal{V}| = 10$
- What is the perplexity of this data for a model that assigns $p(\cdot) = \frac{1}{10}$ to each digit

$$
\begin{aligned}
PP &= 2^{-\frac{1}{m}\sum_{i=1}^{M} \log_2 (\frac{1}{10})^{|\bar{x}^{(i)}|}} \\
&= 2^{-\frac{1}{m}\sum_{i=1}^{M} |\bar{x}^{(i)}| \log_2 \frac{1}{10}} \\
&= 2^{-\log_2 \frac{1}{10}} = 2^{-\log_2 10^{-1}} = 10
\end{aligned}
$$

- Perplexity is weighted equivalent **branching factor**

Evaluation

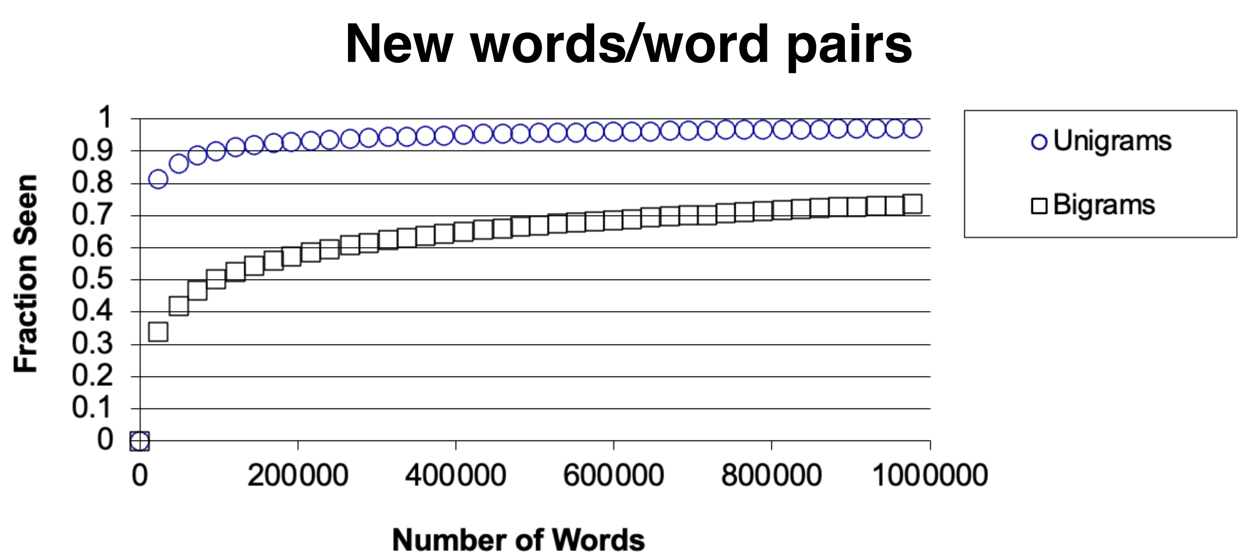
Perplexities of Contemporary Models



https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Sparsity in Language Models

- N-gram models work well if test data is looks like training corpus
  - This is rarely the case, so need models that generalize
- Specifically, with n-gram models: new n-grams appear all the time
  - New words too! More on that a bit later
- This means encountering zeros during test



Sparsity in Language Models

Zeros

Training Set	Test Set
… denied the allegations	… denied the offer
… denied the reports	… denied the loan
… denied the claims	
… denied the request	

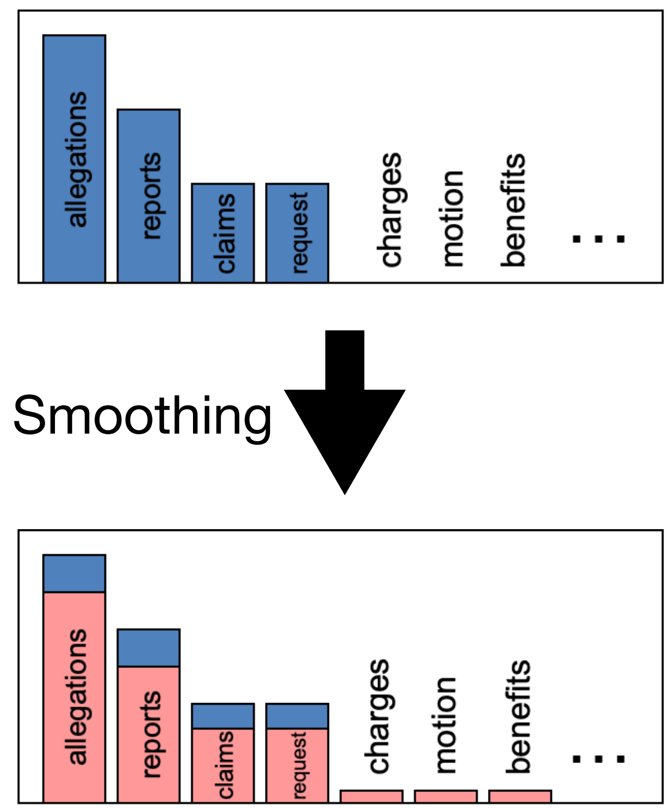
$$p(\text{offer} \,|\, \text{denied the}) = 0$$

- A single n-gram with zero probability → the probability of the entire test set is zero
- Can't even compute perplexity (can't divide by zero)

Smoothing

Intuition

- Estimating statistics from sparse data
- Smoothing steals probability mass to generalize better
- Very important across NLP, but easy to do badly
- Not gone in neural models, just implicit



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

7 total

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

7 total

Smoothing

Add-one Estimation

- Pretend we saw each word one more time than we did
- So, just add one to all counts
  - And don't forget to adjust normalization properly

$$p_{\mathrm{MLE}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i)}{c(x_{i-1})} \longrightarrow p_{\mathrm{Add-1}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + |\mathcal{V}|}$$

- Also called Laplace smoothing

Smoothing

Generalizing Add-one Smoothing

- Add-k:

$$p_{\text{Add-k}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + k}{c(x_{i-1}) + k\,|\mathcal{V}\,|}$$

- Unigram prior smoothing:

$$p_{\text{Add-U}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + mp(x_i)}{c(x_{i-1}) + m}$$

Berkeley Restaurant Corpus

- 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

Berkeley Restaurant Corpus

Raw Counts (9222 sentences)

- Bigrams

	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

- Unigram

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

Berkeley Restaurant Corpus

Normalized Bi-gram Probabilities

$$p_{\text{MLE}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i)}{c(x_{i-1})}$$

	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

Berkeley Restaurant Corpus

Counts with Add-one Smoothing

	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

Berkeley Restaurant Corpus

Add-one Smoothed Bi-gram Probabilities

$$p_{\text{Add}-1}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + |\mathscr{V}|}$$

	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

Berkeley Restaurant Corpus

Reconstituted Counts

$$p_{\text{Add}-1}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + V}$$

$$c^*(x_{i-1}, x_i) = \frac{(c(x_{i-1}, x_i) + 1)c(x_{i-1})}{c(x_{i-1}) + V}$$

	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

Berkeley Restaurant Corpus

Original vs. Reconstituted 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-one Smoothing

- Simple, but very blunt instrument
- In practice, a relatively poor choice for n-gram language models
- But can be useful in domains where the number of zeros doesn't dominate

Smoothing

Backoff and Linear Interpolation

- Sometimes it helps to use **lower-order** n-grams
  - Condition on less context, means you are more likely to have stronger support from the training data (i.e., more common event)
- **Backoff:** use lower-order n-gram
  - For tri-gram, use tri-gram if you have good evidence, otherwise use bi-gram, otherwise unigram
- **Linear interpolation:** mix lower-order n-grams
  - For tri-gram, mix with with bi-gram and unigram probabilities
- Interpolation works better

Linear Interpolation

- Simple interpolation

$$P_\lambda(x_i \,|\, x_{i-1}, x_{i-2}) = \lambda_3 p_{\text{MLE}}(x_i \,|\, x_{i-1}, x_{i-2}) + \lambda_2 p_{\text{MLE}}(x_i \,|\, x_{i-1}) + \lambda_1 p_{\text{MLE}}(x_i)$$

$$\sum \lambda_i = 1$$

- Lambdas conditioned on context

$$\begin{aligned} P_\lambda(x_i \,|\, x_{i-1}, x_{i-2}) = {} & \lambda_3\binom{x_{i-1}}{x_{i-2}} p_{\text{MLE}}(x_i \,|\, x_{i-1}, x_{i-2}) + \\ & \lambda_2\binom{x_{i-1}}{x_{i-2}} p_{\text{MLE}}(x_i \,|\, x_{i-1}) + \\ & \lambda_1\binom{x_{i-1}}{x_{i-2}} p_{\text{MLE}}(x_i) \end{aligned}$$

- Are these well defined distributions?

Linear Interpolation

How to Set the Lambdas?

- Use a held-out corpus
- Choose λs to maximize the probability of the held-out data
  - Fix MLE n-gram probabilities (on training data)
  - Then search over λ values to maximize the probability of the held-out data

Smoothing

Advanced Methods

- General intuition: use the counts of rare events to estimate the probability of events we haven't seen

- Used by many smoothing algorithms

  - Good-Turing

  - Knesser-Ney

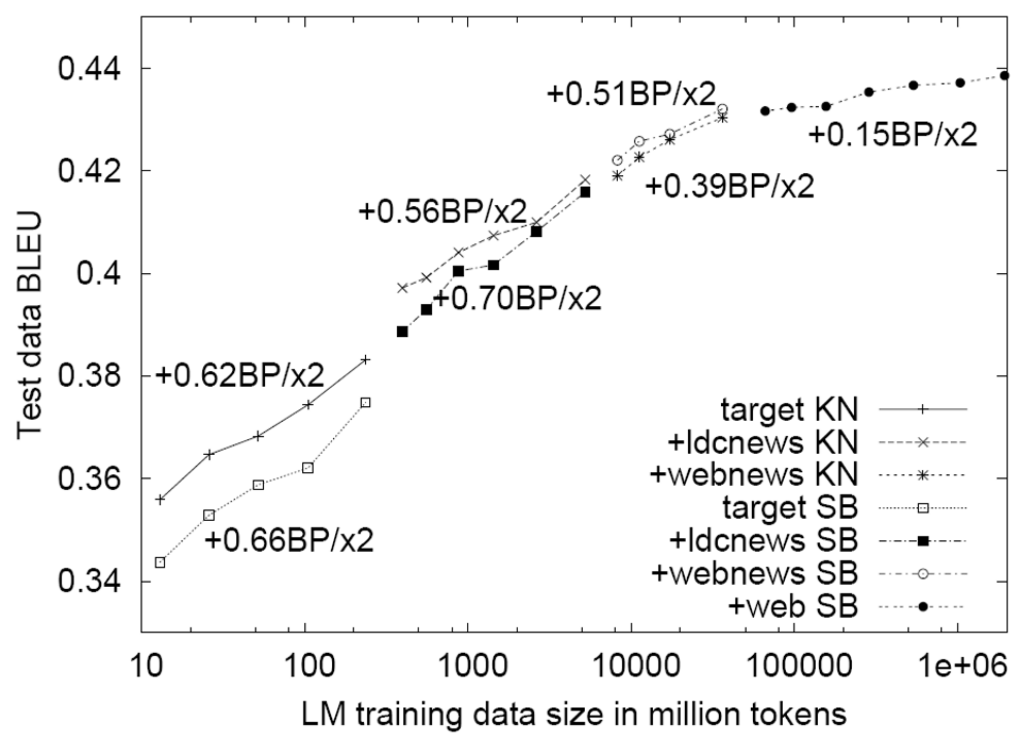
  - Witten-Bell

Smoothing

Data Scale vs. Method

- Having more data is better, and techniques that scale win
- But requires crazy scaling tricks
  - Pruning to only store estimates we trust
  - Efficient data structures (e.g., tries)
  - Bloom filters for approximate language models
  - Storing words as indexes, not strings
  - Using Huffman code for efficient index assignment
  - Quantize probabilities

Extrinsic evaluation using phrase-based machine translation



http://www.aclweb.org/anthology/D07-1090.pdf

Unknown Words

- If we know all the words in advance, vocabulary $\mathscr{V}$ is fixed → closed vocabulary task
- This is rare and unlike, and often, we can't tell, and we have open vocabulary tasks
- Out of vocabulary = OOV words
- A lot of room for creativity around handling OOVs

Unknown Words

The Most Basic Approach

- Create an unknown word token <UNK>
- Training of <UNK> probabilities
  - Create a fixed lexicon L (e.g., rare words are not in L)
  - At text normalization phase, any training word not in L changed to <UNK>
  - Now we estimate probabilities like a normal word
- At decoding time
  - Normalize and use UNK probabilities for any word not in training

Acknowledgements

- Slides adapted from or inspired by Dan Klein, Dan Jurafsky, and Luke Zettlemoyer