Lect 6 N-gram Language Modeling

Why Probabilistic Language Models

Goal: assign a probability to a sentence ("as used by native speakers")

Why do we need probabilistic language models?

Machine Translation: to generate better translations

P(**high** winds tonite) **>** P(**large** winds tonite)

Spell Correction: to the much more likely to happen(i.e., more correct)

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: given a corpus, compute the probability of a sentence W (or sequence of words w_1 w_2 w_3 w_4 w_5...w_n):

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

P(How to cook rice) = P(How, to, cook, rice)

Related task: probability of an upcoming word. That is, given the sentence (w_1,w_2,w_3,w_4), what is the probability that w_5 will be the next word:

$P(w_5|w_1,w_2,w_3,w_4)$ //P(rice| how, to, cook)

related to $P(w_1,w_2,w_3,w_4,w_5)$ //P(how, to, cook, rice)

A model that computes:

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

Better: **The grammar = language model**

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

Reminder: The Chain Rule

Recall the definition of conditional probabilities:

$$P(A|B) = \frac{P(A,B)}{P(B)} \quad \text{Rewriting} \Rightarrow \quad P(A|B) \times P(B) = P(A,B)$$

$$\text{or } P(A,B) = P(A|B) \times P(B)$$

More variables:

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

Example: P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)× P(so|its water is)× P(transparent | its water is so)

The Chain Rule in general is:

$$P(w_1,w_2,w_3,\ldots,w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1,w_2) \times \ldots \times P(w_n|w_1,\ldots,w_{n-1})$$

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

How to Estimate Probabilities

Given a large corpus of English (that represents the language), should we just divide all words and count all probabilities?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

No! Too many possible sentences!
We'll never have enough data (the counts of all possible sentences) for estimating these.

Markov Assumption

Based on [2]

Instead, we apply a simplifying assumption:

Andrei Markov
(1856–1922),
Russian mathematician

Markov suggests: Instead of the counts of all possible sentences **it is enough to only count the last few words**

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

In other words, approximate each component in the product (this is enough)

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

Example:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

Or maybe better:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Unigram Model -Simplest case of Markov Model

Estimate the probability of whole sequence of words by the product of probability of individual words:

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

P(its water is so transparent that the) ≈
P(its) x P(water) x P(is) x P(so) x P(transparent) x P(that) x P(the)

Example of some automatically generated sentences from a unigram model, (words are independent):

```
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
```

➔ This is not really a useful model

Bigram Model

Condition on the previous word:

Estimate the probability of a word given the entire prefix (from the begging to the pervious word) only by the pervious word.

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

P(its water is so transparent that the) ≈ P(water | its) x P(is | water))
x P(so | is) x P(transparent | so) x P(that | transparent) x P(the | that)

➔ The used conditioning (bigram) is still producing something is wrong/weak!

N-gram models

We can extend to 3-grams, 4-grams, 5-grams

In general this is an insufficient model of language

– because language has **long-distance dependencies**:

Predict: "the computer crashed"!!

"**The computer** which I had just put into the machine room on the fifth floor **crashed.**"

This means that we have to consider lots of long sentences.

But in practice we can often get away with N-gram model.

Estimating Bigram Probabilities

The Maximum Likelihood Estimate

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

if we have word w_{i-1}, how many times it was followed by word w_i

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

Example:

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

Sample Corpus

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

Another Example

Given this larger 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 …

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

Many counts are Zero

Raw bigram probabilities

Normalizing the previous table/counts with the following:

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

What kinds of knowledge?

P(english | want) = .0011

P(chinese | want) = .0065

P(to | want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

➔ These numbers reflect how English is used in practice (**our corpus**).

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!

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

How to know a language is model is good?

Does the language model prefer good sentences to bad ones?

- Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences?

Train parameters of our model on a **training set**.

Test the model’s performance on data you 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.

➔ Two way to evaluate a language model

- **Extrinsic** evaluation (**in-vivo**)
- **intrinsic** evaluation (**perplexity)**

Extrinsic evaluation of N-gram models

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 words translated correctly
- Compare accuracy for A and B

➔ Extrinsic evaluation is time-consuming; can take days or weeks

Intuition of Perplexity (intrinsic evaluation)

– 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 ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

A better model of a text

– is one which assigns a higher probability to the word that actually occurs, gives, Gives the highest P(sentence).

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

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

Minimizing perplexity is the same as maximizing probability

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"

$Perplexity_A(T) = P_A(red\ red\ red\ red\ blue)^{-1/5} = ((⅓)^5)^{-1/5} = (⅓)^{-1} = 3$

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:

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

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

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

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)!

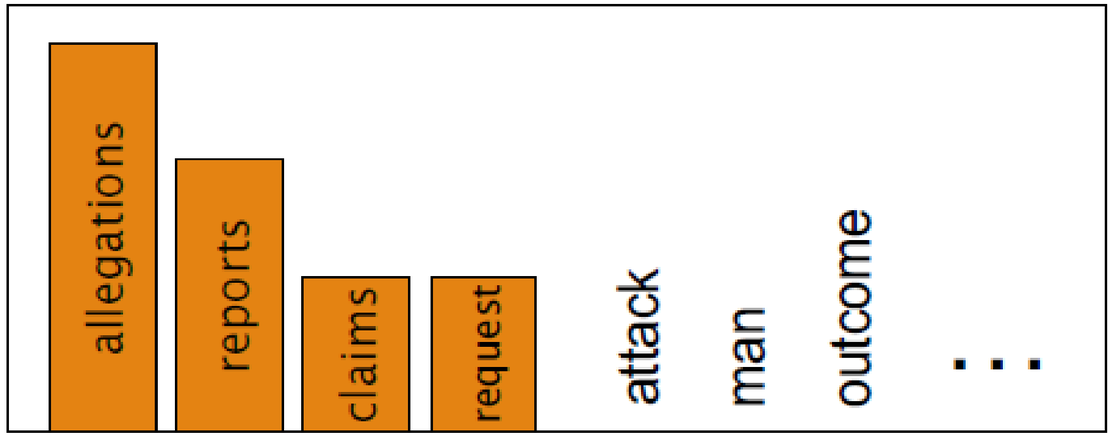
N-gram Language Modeling

Smoothing, Interpolation, and Backoff

The intuition of smoothing (from Dan Klein)

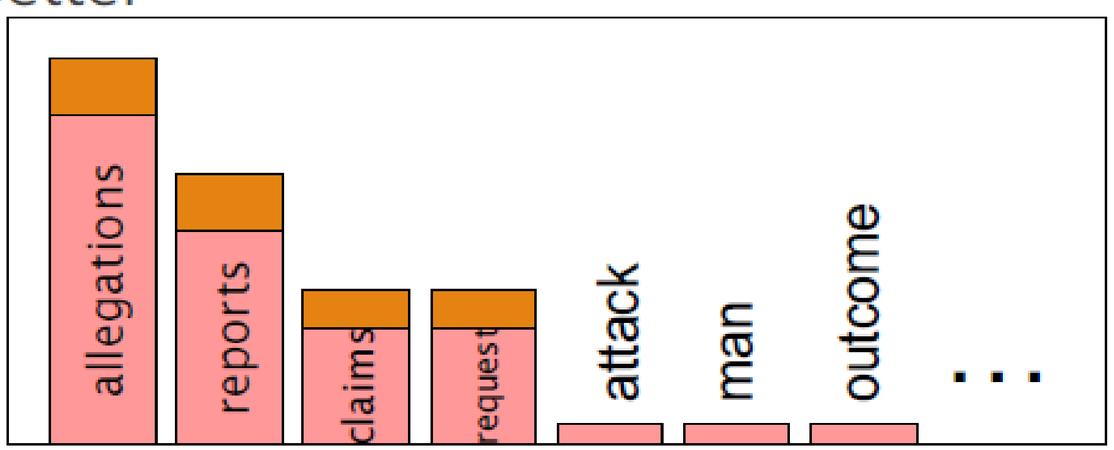
When we have sparse statistics:

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:

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

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

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

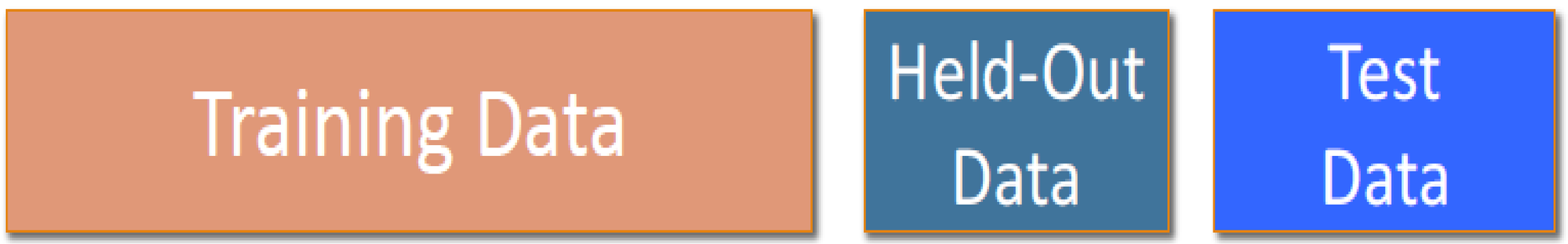
Simple interpolation

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

$$\sum_i \lambda_i = 1$$

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)