Language Models

CS4742 Natural Language Processing
Lecture 01

Jiho Noh

Department of Computer Science
Kennesaw State University

CS4742 Summer 2025 [1]

[1] This lecture is based on the slides from Dr. Hafiz Khan at KSU.

New York Times

- New York Times
- Business, 9/5/19
- “A.I. Can Now Handle This 8th Grade Test. Can You?”
- In New York State, there is a greater chance of precipitation falling as snow in January than in March, because in January the Northern Hemisphere is titled
  - Toward the Sun, and temperatures are warmer
  - Toward the Sun, and temperatures are colder
  - Away from the Sun, and temperatures are warmer
  - Away from the Sun, and temperatures are colder

Next Word Prediction

- From a NY Times story...
  - Stocks plunged this ...
  - Stocks plunged this morning, despite a cut in interest rates by ...
  - Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall ...
  - Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began

Human Word Prediction

- Clearly, at least some of us have the ability to predict future words in an utterance. How?
  - Domain knowledge
  - Syntactic knowledge
  - Lexical knowledge

Word Prediction

- A useful part of the knowledge needed to allow Word Prediction can be captured using simple statistical techniques.
- We'll rely on the notion of the probability of a sequence (of letters, words,...).
- Formula: $P(w_n|w_1 w_2 w_3 \ldots w_{n-1})$

N-Gram Models of Language

- Markov assumption: Use the previous N-1 words in a sequence to predict the next word
- Language Model (LM)
  - unigrams, bigrams, trigrams, ...
- How do we train these models?
  - Very large corpora
- Corpora are online collections of text and speech
  - Brown Corpus
  - Wall Street Journal
  - AP newswire

Language Model (LM)

- **LM:** Assign a probability to a sentence
  - Why?
  - Applications - machine translation, spell correction, speech recognition, etc.
- **Machine Translation:** return text in the target language
  - $P(\text{high winds tonight}) > P(\text{large winds tonight})$
- **Spell Correction:** return corrected spelling of input
  - The office is about fifteen minuets from my house
  - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
- **Speech Recognition:** return a transcript of what was spoken
  - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
- + Summarization, question-answering, etc., etc.!!

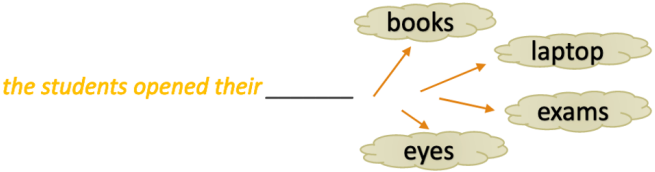
We can use a language model to score possible output strings so that we can choose the best (i.e., most likely) one: if $P(A) > P(B)$, return A, not B.

Language Model

- **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**.



How to Compute Probability

- How to compute this joint probability:

$$P(\text{its, water, is, so, transparent, that})$$

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

Chain Rule

- Recall the definition of conditional probabilities
  - Conditional probability of an event B is the probability that the event will occur given the knowledge that an event A has already occurred.
  - $P(A, B) = P(B|A) \cdot P(A)$
- More variables:
  - $P(A, B, C, D) = P(A) \cdot P(B|A) \cdot P(C|A, B) \cdot P(D|A, B, C)$
- The Chain Rule in General:
  - $P(x_1, x_2, x_3, \ldots, x_n) = P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_1, x_2) \cdots P(x_n|x_1, \ldots, x_{n-1})$

Chain Rule

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

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

Example

$$\begin{aligned} P(\text{“its water is so transparent”}) = P(\text{its}) &\times P(\text{water}|\text{its}) \\ &\times P(\text{is}|\text{its water}) \\ &\times P(\text{so}|\text{its water is}) \\ &\times P(\text{transparent}|\text{its water is so}) \end{aligned}$$

How to estimate these probabilities

- Could we just count and divide?
- Estimate $P(\text{the} \mid \text{its water is so transparent that}) = ?$

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

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Estimate Probabilities

- Markov Assumption/condition: The probability of a word depends only on the previous word. (simplified)
- Simplifying assumption:

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

- Or maybe

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

Estimate Probabilities: Markov Assumption

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

where k is a small number. And we approximate each component in the product:

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

Unigram LM

When $k = 0$, we have a **unigram model**:

$$P(w_1 w_2 w_3 w_4) \approx P(w_1)P(w_2)P(w_3)P(w_4) = \prod_{i=1}^{n} P(w_i)$$

- Some automatically generated sentences 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

Example: Unigram LM

Text: beginning by, very Alice but was and? reading no tired of to into sitting sister the, bank, and thought of without her nothing: having conversations Alice once do or on she it get the book her had peeped was conversation it pictures or sister in, 'what is the use had twice of a book' 'pictures or' to

- P(of) = 3/66
- P(her) = 2/66
- P(Alice) = 2/66
- P(was) = 2/66
- P(to) = 2/66
- P(,) = 4/66
- P(') = 4/66

Why is this a bad model?

Bigram model

- Condition on the previous word:

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

- Generated senteces using 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

N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general, this is an insufficient model of language
  - because language has **long-distance dependencies**:
  - Example: *"The computer which I had just put into the machine room on the fifth floor crashed."*
- But we can often get away with N-gram models

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.
- E.g., MT: select the most likely translated sentence among many possibilities.

N-gram Language Models

- the students opened their _____
- **Question:** How do we learn a language model?
- **Answer (before deep learning):** use n-gram language model
- **Definition:** a n-gram is a sequence of n-consecutive words
- **Example:**
  - 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"
- Estimating bigram probabilities: Collect the statistics about the frequency of the n-gram in some corpus, and use it to estimate the probability

The Maximum Likelihood Estimate

- The Maximum Likelihood Estimate

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

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

$P(\text{I}|\text{<s>}) = 2/3 = .67$ $P(\text{Sam}|\text{<s>}) = 1/3 = .33$ $P(\text{am}|\text{I}) = 2/3 = .67$
$P(\text{</s>}|\text{Sam}) = 1/2 = .50$ $P(\text{Sam}|\text{am}) = 1/2 = .50$ $P(\text{do}|\text{I}) = 1/3 = .33$

Practice Example

Let's calculate the probability of the word "like" occurring after the word "really".

- Thank you so much for your help.
- I really appreciate your help.
- Excuse me, do you know what time it is?
- I'm really sorry for not inviting you.
- I really like your watch.

$$P(\text{like} \mid \text{really}) = ?$$

More Examples:

- Berkeley Restaurant Project sentences
- Total 9222 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
  - ...

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

Compute 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

$$
\begin{aligned}
&P(\langle s \rangle \text{ I want english food } \langle /s \rangle) \\
&= P(\text{I}|\langle s \rangle) \times P(\text{want}|\text{I}) \times P(\text{english}|\text{want}) \times P(\text{food}|\text{english}) \times P(\langle /s \rangle|\text{food}) \\
&= 0.000031
\end{aligned}
$$

What kinds of knowledge?

- $P(\text{english}|\text{want}) = 0.0011$
- $P(\text{chinese}|\text{want}) = 0.0065$
- $P(\text{to}|\text{want}) = 0.66$
- $P(\text{eat}|\text{to}) = 0.28$
- $P(\text{food}|\text{to}) = 0$
- $P(\text{want}|\text{spend}) = 0$
- $P(\text{i}|\text{<s>}) = 0.25$

Computing Issues

- We do everything in log space
- Avoid underflow
- (also adding is faster than multiplying)

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

Google n-gram model

- https://ai.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
```

3-gram data

```
ceramics collectables collectibles 55
ceramics collectables fine 130
ceramics collected by 52
ceramics collectible pottery 50
ceramics collectibles cooking 45
ceramics collection , 144
ceramics collection . 247
ceramics collection </S> 120
ceramics collection and 43
ceramics collection at 52
ceramics collection is 68
```

Example: Trigram Model

- Build LM using NLTK Reuters corpus
- A collection of 10,788 news documents
- Totaling 1.3 million words

```
1  from nltk.corpus import reuters
2  from nltk import bigrams, trigrams
3  from collections import Counter, defaultdict
4
5  # Create a placeholder for model
6  model = defaultdict(lambda: defaultdict(lambda: 0))
7
8  # Count frequency of co-occurance
9  for sentence in reuters.sents():
10     for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
11         model[(w1, w2)][w3] += 1
12
13 # Let's transform the counts to probabilities
14 for w1_w2 in model:
15     total_count = float(sum(model[w1_w2].values()))
16     for w3 in model[w1_w2]:
17         model[w1_w2][w3] /= total_count
```

Example: Trigram Model

```python
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for sentence in reuters.sents():
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        model[(w1, w2)][w3] += 1

# Let's transform the counts to probabilities
for w1_w2 in model:
    total_count = float(sum(model[w1_w2].values()))
    for w3 in model[w1_w2]:
        model[w1_w2][w3] /= total_count
```

Output

```python
sorted(dict(model["the","news"]).items(), key=lambda x:-1*x[1])
```

```
[('conference', 0.25),
 ('of', 0.125),
 ('.', 0.125),
 ('with', 0.08333333333333333),
 (',', 0.08333333333333333),
 ('agency', 0.08333333333333333),
 ('that', 0.08333333333333333),
 ('brought', 0.041666666666666664),
 ('about', 0.041666666666666664),
 ('broke', 0.041666666666666664),
 ('on', 0.041666666666666664)]
```

Model Evaluation

- 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

- 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

- Difficulty of extrinsic evaluation:
  - Time-consuming; can take days or weeks
  - 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.

Intrinsic Evaluation - Perplexity

The Shannon Game:

- 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 _____
- Utilize the pre-computed probability distribution over words
- Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

Perplexity

- The best language model is one that best predicts an unseen test set
  - Gives the highest P(sentence)
- Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} = \left(\prod_{i=1}^{N} P(w_i | w1, w2, \ldots, w_{i-1}) \right)^{-\frac{1}{N}}$$

For bigram:

$$PP(W) = \left(\prod_{i=1}^{N} P(w_i | w_{i-1}) \right)^{-\frac{1}{N}}$$

Minimizing perplexity is equivalent to maximizing the n-gram language model probability

Perplexity

- Let's suppose a sentence consisting of random digits.
- What is the perplexity of this sentence according to a model that assigns $P = \frac{1}{10}$ to each digit?

Perplexity

- Let's suppose a sentence consisting of random digits.
- What is the perplexity of this sentence according to a model that assigns $P = \frac{1}{10}$ to each digit?
- solution:

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

Perplexity – Example

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

The Shannon Visualization Method

1. Choose a random bigram (<s>, w) according to its probability
2. Now choose a random bigram (w, x) according to its probability
3. And so on until we choose </s>
4. 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
```

Example — Approximate 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

Issues with n-gram

- Suppose we are learning a 4-gram language model.
  - Example: as the proctor started the clock, the students opened their _____
- For example, suppose that in the corpus:
  - “students opened their” occurred 1000 times
  - “students opened their books” occurred 400 times
  - $P(\text{books} \mid \text{students opened their}) = 0.4$
  - "students opened their exams" occurred 100 times
  - $P(\text{exams} \mid \text{students opened their}) = 0.1$

Should we have discarded the “proctor” context?

Issues with n-gram

- 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.
- 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)!

Issues with n-gram — example

Training set:

- . . . denied the allegations
- . . . denied the reports
- . . . denied the claims
- . . . denied the request

Test set:

- . . . denied the offer
- . . . denied the loan

Probability:

$$P(\text{“offer”} \mid \text{denied the}) = 0$$

Storage Problems With N-gram Language Models

- Increasing n or increasing corpus increases model size!
- Count(students opened their w)

Uni-gram (n = 1)	$P(w_i)$	N parameters
Bi-gram (n = 2)	$P(w_i \mid w_{i-1})$	N^2 parameters
Tri-gram (n = 3)	$P(w_i \mid w_{i-2}w_{i-1})$	N^3 parameters
Four-gram/quadrigram* (n = 4)	$P(w_i \mid w_{i-3}w_{i-2}w_{i-1})$	N^4 parameters

$$\text{P(w} \mid \text{ students opened their)} = \frac{\text{Count(students opened their } w\text{)}}{\text{Count(students opened their)}}$$

Need to store count for all the n-gram for your corpus

Storage Problems With N-gram Language Models

	Large n	Small n
Pros	More context, helps predicting the next word	fewer parameters, inexpensive to estimate, more reliable parameters even with smaller corpus
Cons	exponentially more parameters, computationally expensive, require a large corpus to estimate parameters (still likely unreliable due to sparsity)	Lack context for predicting next word

Issues with n-gram: Sparsity Problems

- Use larger n-grams makes predictions more accurate but causes the sparsity problems

$$\mathrm{P}(\mathrm{w} \mid \text{students opened their}) = \frac{\text{Count(students opened their } w)}{\text{Count(students opened their)}}$$

- **Problem:** What if "students opened their w" never occurred in data? Then w has probability 0!
  - **Partial Solution:** Add small δ to the count for every w in the vocabulary (V). This is called **smoothing**.
- **Problem:** What if "students opened their" never occurred in data? Then we can't calculate probability for any w!
  - **Partial Solution:** Just condition on "opened their" instead. This is called **backoff**.

Back-off

- In given corpus, we may have never seen
- Smoothing gives same probability for these two sentences:
  - Scottish beer *drinkers*
  - Scottish beer *eaters*
- We may have seen “beer drinker” more often than “beer eater”
- Better: back-off to bigram

Interpolation

- Higher and lower order n-gram models have different strengths and weaknesses
  - High-order n-grams are sensitive to more context, but have sparse counts
  - Low-order n-grams consider only very limited context, but have robust counts
- Combine them:

$$P_I(w_3|w_1, w_2) = \lambda_1 P(w_3|w_1, w_2) + \lambda_2 P(w_3|w_2) + \lambda_3 P(w_3)$$

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:

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

Add-1 estimate:

$$P_{Add-1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + V}$$

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 $400/1,000,000 = 0.0004$.
- 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: Raw 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

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

$$P^*(w_{n-1}w_n) = \frac{[C(w_{n-1}, w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	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

Reconstituted counts — Observation

	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

- C(want to) went from 608 to 238, P(to | want) from .66 to .26!
- Discount $d = \frac{c^*}{c}$, d for "chinese food" = .10! A 10x reduction

Reconstituted counts — Motivation

$$P^*(w_{n-1}w_n) = \frac{[C(w_{n-1}, w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

- To improve the estimation of probabilities for rare or unseen events by redistributing probability mass from more frequent events,
- thereby enhancing the model's ability to generalize from limited data.

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

$$\hat{P}(w_3|w_1, w_2) = \lambda_1 P(w_3|w_1, w_2) + \lambda_2 P(w_3|w_2) + \lambda_3 P(w_3)$$

where $\sum_i \lambda_i = 1$.

We can condition lambda's on the context

How to set the lambdas?

- Use a held-out corpus

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

$$\log P(w_1, \ldots, w_n | M(\lambda)) = \sum_{i=1}^{n} \log P_M(\lambda)(w_i | w_{i-1})$$

Open versus Closed Vocabulary Tasks

- 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
- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - ★ Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)

Summary

Questions?