# Introduction to Language Modeling

Manu Madhavan

Department of Computer Science and Engineering,
Amrita School of Engineering

Topic 6

# Outline

- Why probability to explain theory of language?
- Language Modeling
- Basic idea
- Examples
- N-gram Language Modeling

# Language and cognition as probabilistic phenomena

- Human cognition is probabilistic and that language must therefore be probabilistic too since it is an integral part of cognition.
- We live in a world filled with uncertainty and incomplete information.
- We make decisions by processing these information
- These cognitive processes are best formalized as probabilistic processes
- Processing the words, forming an idea of the overall meaning of the sentence, and weighing it in making a decision is no different in principle
- Meaning of the word is defined by the circumstances of its use $\rightarrow$ uncertainty
- So Probability plays an important role in tackle the questions of meaning

# Language Modeling

- A language model is a probability distribution over strings on an alphabet

# Language Modeling

- A language model is a probability distribution over strings on an alphabet
- Language Models are the development of probabilistic models that are able to predict the next word in the sequence given the words that precede it.

# Language Modeling

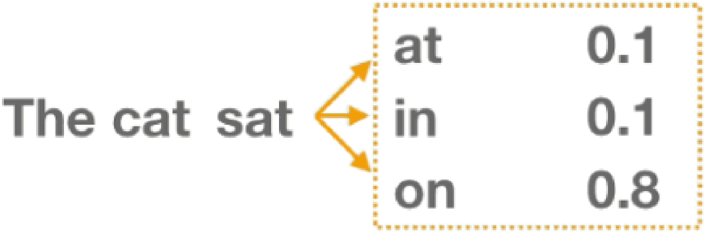
- A language model is a probability distribution over strings on an alphabet
- Language Models are the development of probabilistic models that are able to predict the next word in the sequence given the words that precede it.
- Language models explain various linguistics phenomena using probability
- It is a root problem for a large range of natural language processing tasks
  - Machine Translation
  - Spell checking
  - Next word prediction
  - Text generation

# Example: Word Prediction

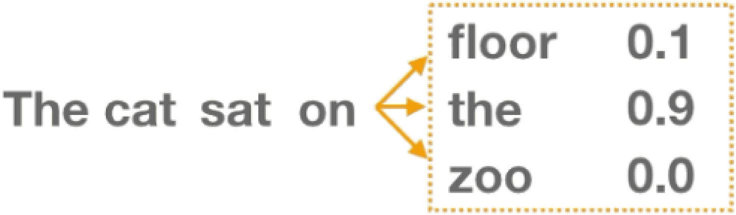
The cat

# Example: Word Prediction

# Example: Word Prediction



# Example: Word Prediction



# Example: Word Prediction

# Language Model- Other Applications

- LM can assign probability to the sentence (how probable the sequences are?)
  $P(The\ cat\ sat\ on\ the\ mat) =$
  $P(The).P(cat|the).P(sat|the\ cat).P(on|The\ cat\ sat)...$
- This will give higher probability to common/syntactically/semantically correct usages
  Large wind tonight $\rightarrow$ 0.31
  High wind tonight $\rightarrow$ 0.97

# Language Model-Other Applications

- Spelling mistake or correct word?
  $P(I\ was\ late\ by\ fitfteen\ minuets) > P(I\ was\ late\ by\ fiteen\ minutes)$
- Speech Recognition
  $P(I\ saw\ a\ van) >> P(eyes\ awe\ of\ an)$
- Machine Translation: which usage is more plausible in target language
  $P(high\ wind) > P(large\ wind)$
  $P(He\ immediately\ moved\ to\ hospital) >$
  $P(He\ quickly\ moved\ to\ hospital$

# Language Model-Other Applications

- Spelling mistake or correct word?
  $P(I\ was\ late\ by\ fitfteen\ minuets) > P(I\ was\ late\ by\ fiteen\ minutes)$
- Speech Recognition
  $P(I\ saw\ a\ van) >> P(eyes\ awe\ of\ an)$
- Machine Translation: which usage is more plausible in target language
  $P(high\ wind) > P(large\ wind)$
  $P(He\ immediately\ moved\ to\ hospital) >$
  $P(He\ quickly\ moved\ to\ hospital$

# Language Model-Other Applications

- Context sensitive spelling correction
- Natural Language generation
- Next word prediction
- Auto completion (In mail, search engines)
  - Please, call .......
  - Can you .... .... ....

# Language Model

- Goal: Compute the probability of a sentence or sequence of words $P(W) = P(w_1, w_2, ....w_n)$
- Related Task:
  $P(w_{n+1}|w_1, ...w_n)$
- Language model compute either of the above

# Language Model

- Probability of a sentence is the joint probability of the words
  $P(W) = P(w_1, w_2, ....w_n)$
- This relay on conditional probability
  $P(A|B) = \frac{P(A,B)}{P(B)}$
  $P(A, B) = P(B)P(A|B)$
- This can be expanded
  $P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)$

# Counting Probabilities

- $P(the\ cat\ sat\ on\ the\ mat) =$
  $P(the)P(cat|the)P(sat|the\ cat)...P(mat|the\ cat\ sat\ on\ the)$
- How do you compute this probability?
  $P(sat|the\ cat) = \frac{Count(the,cat,sat)}{Count(the,cat)}$
- We may never see enough data to estimate the long sequences

# Markov Assumption and Language Model

- In the previous examples, the next word is represented by a conditional probability of previous word
  $P(W_{n+1}|W_1...W_n) = \frac{P(W_1....W_{n+1})}{P(W_1....W_n)}$
- Exponential number of sequences, counting is difficult
- The question here is how much history is required?
  - It depends only a limited history (3 or 4 words)
- **Markov Assumption**
  $P(W_{n+1}|W_1...W_n) = P(W_{n+1}|W_n)$
- Generally, we can say, it depends only on previous k words (k-order Markov model, $k < n$)
  $P(W_{n+1}|W_1...W_n) = P(W_{n+1}|W_{n-k}...W_n)$

# Markov Assumption and n-gram model

Looks only at n words at a time.

- $0^{th}$ order: $P(W_{n+1}|W_1...W_n) = P(W_{n+1})$ (independent of previous words, unigram model)
- $1^{st}$ order: $P(W_{n+1}|W_1...W_n) = P(W_{n+1}|W_n)$ (bigram model) $P(cat|the)$, $P(sat|cat)$..
- $2^{nd}$ order: $P(W_{n+1}|W_1...W_n) = P(W_{n+1}|W_n, W_{n-1})$ (trigram model)

# Simple N-Gram Model

- Goal is to estimate the probability of a word given history, $P(w|h)$
- $P(apple|the\ \ boy\ \ ate)$
- $P(apple|the\ \ boy\ \ ate) = \frac{Count(the\ \ boy\ \ ate\ \ apple)}{Count(the\ \ boy\ \ ate)}$
- Beware!— language is creative and new sentences are added every time
- It is hard to estimate sufficient statistics even from a large corpus (like web)

# Some Notations

- Word sequence $w_1, ...w_n$ is represented as $w_1^n$
- Joint probability $P(X = w_1, Y = w_2, ....)$ is represented as $P(w_1, w_2, ...w_n)$

# Chain Rule

- $P(X_1, ...X_n) = P(X_1)P(X2|X_1)P(X_3|X_1, X_2)....P(X_n|X_1, ..X_{n-1})$
- $P(X_1, ...X_n) = \prod_{i=1}^{n} P(X_i|X_1^{i-1})$
- $P(w_1^n) = \prod_{i=1}^{n} P(w_i|w_1^{i-1})$
- The joint probability of a sequence and computing the conditional probability of a word given previous words
- We don't know any way to compute the exact probability of a word given a long sequence of preceding words
- Approximate: The intuition of the N-gram model is that instead of computing the probability of a word given its entire history, we will approximate the history by just the last few words.

# Bigram

- Bigram model approximates the probability of a word given all the previous words
- $P(w_n|w_1^{n-1}) = P(w_n|w_{n-1}$
- $P(apple|the\ boy\ ate) = P(apple|ate)$
- Markov Assumption
- We can generalize bigram to **N-gram** model

# How to estimate N-gram probability

- Maximum Likelihood Estimate (MLE)
- For example to compute a particular bigram probability of a word y given a previous word x, we'll compute the count of the bigram C(xy) and normalize by the sum of all the bigrams that share the same first word x
- $P(w_n|w_{n-1}) = \frac{Count(w_{n-1},w_n)}{\sum_w (Count(w_{n-1},w))}$
- Since the sum of all bigram counts that start with a given word $w_{n1}$ must be equal to the unigram count for that word $w_{n1}$
- $P(w_n|w_{n-1}) = \frac{Count(w_{n-1},w_n)}{Count(w_{n-1})}$

# Example

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

Here are the calculations for some of the bigram probabilities from this corpus

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

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

- For N-gram model: $P(w_n|w_1^{n-1}) = \frac{count(w_{n-N+1}^{n-1} w_n)}{Count(w_{n-N+1}^{n-1})}$
- This ratio is called relative frequency

# Example

| | 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 |

**Figure 4.1** Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Fig. 4.2 shows the bigram probabilities after normalization (dividing each row by the following unigram counts):

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

# Example

$$P(\texttt{i} \mid \texttt{<s>}) = 0.25 \qquad P(\texttt{english} \mid \texttt{want}) = 0.0011$$

$$P(\texttt{food} \mid \texttt{english}) = 0.5 \qquad P(\texttt{</s>} \mid \texttt{food}) = 0.68$$

# Example

| | 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 |

**Figure 4.2** Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

$$\begin{aligned}
& P(\texttt{<s> i want english food </s>}) \\
&= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{I})P(\texttt{english}|\texttt{want}) \\
&\qquad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\
&= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
&= = .000031
\end{aligned}$$

# Lecture-11

## So far

- Language Modeling is assigning probabilities to sequence of words/characters
- Markov assumption and N-gram models
- Counting probability
- Example

## Next

- Evaluation - Perplexity
- Smoothing

# Training and Test corpus

- The probabilities of the N-gram model come from the corpus it trained on
- Training on some data and testing on some other data (mutually exclusive sets)
- Divide the dataset into **Training corpus and test corpus**
- Its important not to let the test sentences into the training set

# Training and Test corpus

- The probabilities of the N-gram model come from the corpus it trained on
- Training on some data and testing on some other data (mutually exclusive sets)
- Divide the dataset into Training corpus and test corpus
- **Its important not to let the test sentences into the training set**
- How to split input corpus: 80:20, 70:30,...

# Issues in N-gram modeling

- The probabilities of the N-gram model come from the corpus it trained on
- It may not TRUE for another testing corpus
- Sensitivity to the value of N
- Create a general purpose corpus by collecting different genres

# Treating unknown words

- Test set may contain unknown words (Out of Vocabulary words-OOV)
- Choose a Vocabulary which is fixed
- Convert OOV to $< UNK >$ unknown category
- Estimate the probability of $< UNNK >$

# Evaluating the model

- **Extrinsic** (in vivo) vs intrinsic evaluation
- If know the expected model in prior, we can us extrinsic measures (eg: accuracy)
- For example: We can apply a model on some application (say text classification and predict accuracy)
- **Intrinsic** measures evaluate without any external applications
- Perplexity

# Perplexity

- **Perplexity** is the most common intrinsic evaluation metric for N-gram language models.
- Given two probabilistic models, **the better model is the one that has a tighter fit to the test data**, or predicts the details of the test data better.
- The perplexity (sometimes called PP for short) of a language model on a test set is a function of the probability that the language model assigns to that test set.
- For sequence of $W = w_1, ...w_N$
- $PP(W) = P(w_1, ...1_N)^{-1} = \sqrt[N]{\frac{1}{P(w_1,..w_N)}}$

# Perplexity

- For sequence of $W = w_1, ...w_N$
- $PP(W) = P(w_1, ...1_N)^{-1} = \sqrt[N]{\frac{1}{P(w_1,..w_N)}}$
- $PP(W) = \sqrt[N]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1,..w_{i-1})}}$
- $PP(W) = \sqrt[N]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_{i-1})}}$ For bigram model

# Perplexity


- The best model will be the one that assigns the highest probability to the test set.
- Intuitively, if a model assigns a high probability to the test set, it means that it is not surprised to see it (its not perplexed by it)
- Low perplexity indicates it has a good understanding of how the language works.
- $PP(W) = \sqrt[N]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1,..w_{i-1})}}$ (we are taking inverse of probability)
- A lower perplexity indicates a better model

# Perplexity

- Perplexity also indicate average branching factor of a language model
- Branching factor is the number of possible next words that can follow a word
- perplexity is high means: “*when trying to guess the next word, our model is as confused as if it had to pick between many different words*”

# Next: Smoothing

- The estimating probabilities from sparse data
- Zero count affect entire MLE computation
- Addressing poor estimation: smoothing
- Different Smoothing techniques
  - Laplace smooting
  - Good-Turing discounting
- Interpolation

Read: Chapter 4, SLP, Jurafsky

# Language Modeling

## Smoothing: Add-one (Laplace) smoothing

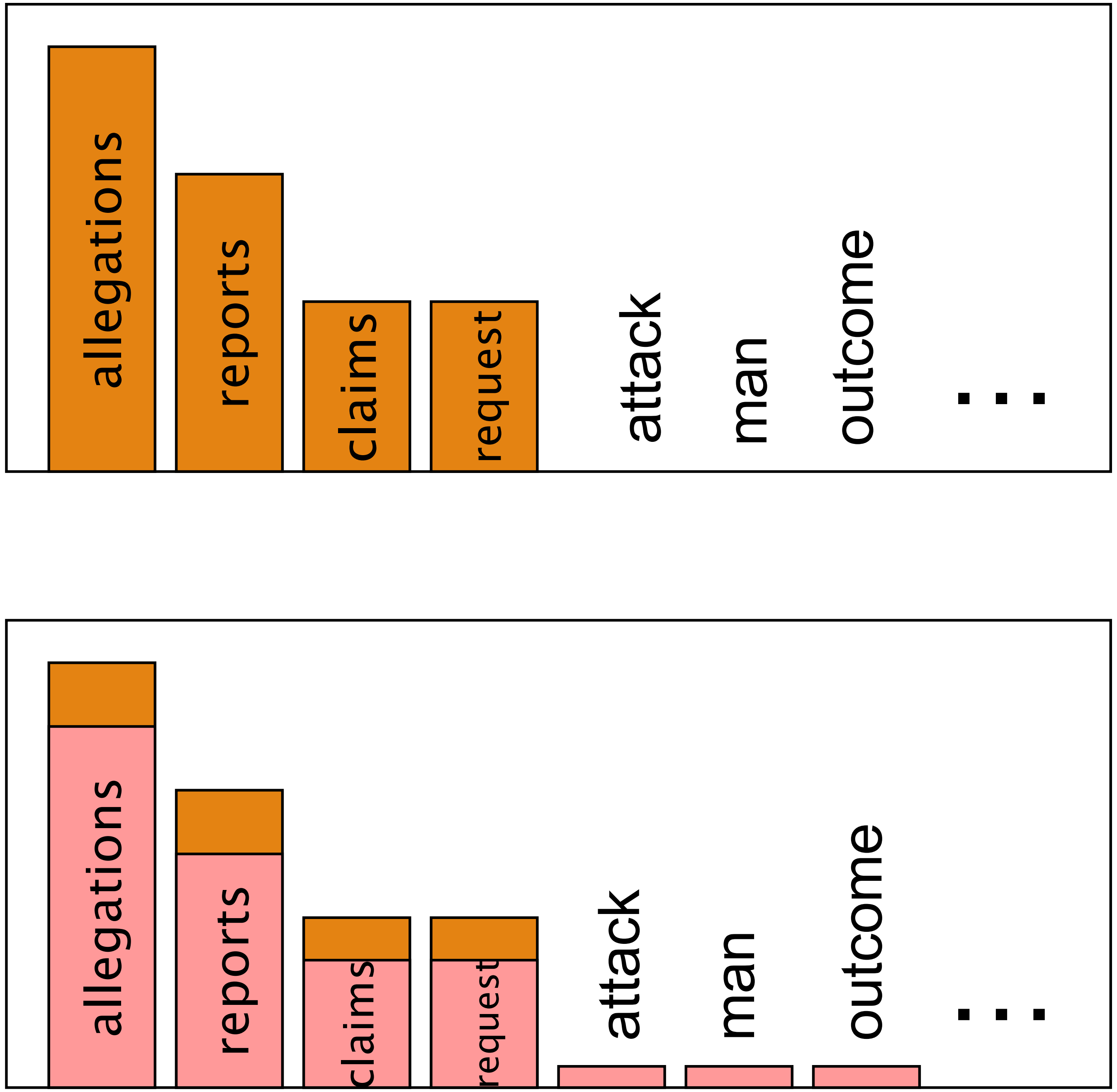
# The intuition of smoothing (from Dan Klein)

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:

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

Add-1 estimate:

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

# 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 = .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:

- We'll see better methods

But add-1 is used to smooth other NLP models

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

# Language Modeling

# Smoothing: Add-one (Laplace) smoothing

# Language Modeling

## Interpolation, Backoff, and Web-Scale LMs

# 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

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

$$\sum_i \lambda_i = 1$$

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 the lambdas?

Use a **held-out** corpus

Training Data

Held-Out Data

Test Data

Choose λs to maximize the 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:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

# Unknown words: Open versus closed vocabulary tasks

If we know all the words in advanced

- 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)

# Smoothing for Web-scale N-grams

"Stupid backoff" (Brants *et al*. 2007)

No discounting, just use relative frequencies

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

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

# N-gram Smoothing Summary

Add-1 smoothing:

- OK for text categorization, not for language modeling

The most commonly used method:

- Extended Interpolated Kneser-Ney

For very large N-grams like the Web:

- Stupid backoff

# Advanced Language Modeling

Discriminative models:

- choose n-gram weights to improve a task, not to fit the training set

Parsing-based models

Caching Models

- Recently used words are more likely to appear

$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2} w_{i-1}) + (1 - \lambda) \frac{c(w \in history)}{\mid history \mid}$$

- These turned out to perform very poorly for speech recognition (why?)

# Language Modeling

## Interpolation, Backoff, and Web-Scale LMs