# N-gram Language Models

**Phạm Quang Nhật Minh**
AImesoft JSC
minhpham0902@gmail.com

February 11, 2023

# Agenda



- Introduction to N-grams
- Estimating N-gram probabilities
- Evaluating language model
- Generalization and zeros
- Smoothing techniques:
  - Add-one (Laplace) smoothing
  - Interpolation, Backoff

# Suggested readings



- Chapter 3. Language Modeling with N-Grams (SLP)
  - https://web.stanford.edu/~jurafsky/slp3/3.pdf
- Language Models, by Michael Collins.
  - http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf
- NLP Programming Tutorials, by Graham Neubig
  - http://www.phontron.com/slides/nlp-programming-en-01-unigramlm.pdf
  - http://www.phontron.com/slides/nlp-programming-en-02-bigramlm.pdf

# The language modeling problem



- Goal: compute the probability of a sentence or sequence of words.

$$P(W) = P(w_1, w_2, \dots, w_n)$$

  - E.g., $P(\text{Hôm nay trời đẹp quá}) = P(\text{Hôm}, \text{nay}, \text{trời}, \text{đẹp}, \text{quá})$

- Related task: probability of an upcoming word:

$$P(w_4 | w_1, w_2, w_3)$$

  - E.g., $P(\text{đẹp} | \text{Hôm}, \text{nay}, \text{trời})$

- A model that computes either of these: $P(W)$ or $P(w_n | w_1, w_2, \dots, w_{n-1})$ is called a language model.

# Why language models?



- Machine Translation:
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - 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)

# How to compute P(W)

- How to compute this joint probability:
  - *P*(its, water, is, so, transparent, that)
- Intuition: let's rely on the Chain Rule of Probability

# The Chain Rule



- Conditional probabilities

$$P(B|A) = P(A,B)/P(A)$$

- Rewriting:

$$P(A,B) = P(A)P(B|A)$$

# The Chain Rule cont.



- More variables:

$$P(A,B,C,D) = ?$$

$$\begin{aligned} P(A,B,C,D) &= P(A,B,C)P(D|A,B,C) \\ &= P(A,B)P(C|A,B)P(D|A,B,C) \\ &= P(A)P(B|A)P(C|A,B)P(D|A,B,C) \end{aligned}$$

# The Chain Rule cont.



- The Chain Rule in general

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

# Applying the Chain Rule for joint probability



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

*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)

# How to estimate these probabilities



- Could we just count and divide?

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

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

# Markov Assumption



- Simplifying assumption:

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

- Or maybe:

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

# Markov Assumption



- We approximate each component in the product

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

- So the joint probability of the sequence is

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

# Simplest case: Unigram model



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

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
```

# Bigram model



- Condition on the previous word:

$$P(w_i|w_1w_2\ldots w_{i-1}) \approx P(w_i|w_{i-1})$$

```
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
```

# Agenda



- Introduction to N-grams
- Estimating N-gram probabilities
- Evaluating language model
- Generalization and zeros
- Smoothing techniques:
  - Add-one (Laplace) smoothing
  - Interpolation, Backoff

# Unigram language model



- Do not use history

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i)$$

- Estimate $P(w_i)$ by using Maximum Likelihood Estimate (MLE)

$$P(w_i) = \frac{\text{count}(w_i)}{\sum_{w'} \text{count}(w')}$$

# Unigram language model: an example



i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

P(nara) = 1/20 = 0.05
P(i) = 2/20 = 0.1
P(</s>) = 3/20 = 0.15

P(W=i live in nara . </s>) =

$0.1 * 0.05 * 0.1 * 0.05 * 0.15 * 0.15 = 5.625 * 10^{-7}$

# Bigram language model



- Condition on the previous word:

$$P(w_i|w_1 w_2 \dots w_{i-1}) \approx P(w_i|w_{i-1})$$

- The Maximum Likelihood Estimate

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

# Bigram language model: an example



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

<s> I am Sam </s>

<s> Sam I am </s>

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

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

$P(</\mathrm{s}> | \mathrm{Sam}) = \frac{1}{2} = .5$ $\quad$ $P(\mathrm{Sam} | \mathrm{am}) = \frac{1}{2} = .5$ $\quad$ $P(\mathrm{do} | \mathrm{I}) = \frac{1}{3} = .33$

# More examples: Berkeley Restaurant Project 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



- Out of 9222 sentences

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 5 | 827 | 0 | 9 | 0 | 0 | 0 | 2 |
| want | 2 | 0 | 608 | 1 | 6 | 6 | 5 | 1 |
| to | 2 | 0 | 4 | 686 | 2 | 0 | 6 | 211 |
| eat | 0 | 0 | 2 | 0 | 16 | 2 | 42 | 0 |
| chinese | 1 | 0 | 0 | 0 | 0 | 82 | 1 | 0 |
| food | 15 | 0 | 15 | 0 | 1 | 4 | 0 | 0 |
| lunch | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| spend | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# Raw bigram probabilities



- Normalize by unigrams:

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

- Results:

| | 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

# Practical Issues



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

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

# Language Modeling Toolkits



- SRILM
  - http://www.speech.sri.com/projects/srilm/
- KenLM
  - https://kheafield.com/code/kenlm/

# Google N-Gram Release, August 2006



AUG
3

## All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

…

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

# Google N-Gram Release



- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

# Google Book N-grams



- http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

# Agenda



- Introduction to N-grams
- Estimating N-gram probabilities
- Evaluating language model
- Generalization and zeros
- Smoothing techniques:
  - Add-one (Laplace) smoothing
  - Interpolation, Backoff

# Evaluation: How good is our model?



- 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

# Evaluation: How good is our model?



- 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.

# Training on the test set



- We can’t allow test sentences into the training set
- We will assign it an artificially high probability when we set it in the test set
- “Training on the test set”
- Bad science!

# Two evaluation approaches



- Extrinsic evaluation
  - Compare two language models in downstream tasks
    - e.g., Spelling correction, speech recognition, MT
- Intrinsic evaluation
  - Use some evaluation measures on the test set
  - We will use **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

# Difficulty of extrinsic evaluation of N-gram models



- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - 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.

# Intuition of 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 ____

    mushrooms 0.1

    pepperoni 0.1

    anchovies 0.01

    ….

    fried rice 0.0001

    ….

    and 1e-100
  - 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

# Likelihood



- Likelihood is the probability of some observed data (the test set $W_{test}$), given the model M

$$P(W_{test}|M) = \prod_{\boldsymbol{w} \in W_{test}} P(\boldsymbol{w}|M)$$

# Log Likelihood



- Likelihood uses very small numbers=underflow
- Taking the log resolves this problem

$$\log P(W_{test}|M) = \sum_{\boldsymbol{w} \in W_{test}} \log(\boldsymbol{w}|M)$$

# Entropy



- Entropy H is average negative $\log_2$ likelihood per word

$$\mathrm{H}(W_{test}|M) = \frac{1}{|W_{test}|} \sum_{\boldsymbol{w} \in W_{test}} -\log_2 P(\boldsymbol{w}|M)$$

- Perplexity is equal to two to the power of per-word entropy

$$\mathrm{PPL} = 2^{H}$$

- Lower perplexity = better model

# Lower perplexity = better model



- Training 38 million words, test 1.5 million words, WSJ

| N-gram Order | Unigram | Bigram | Trigram |
|---|---|---|---|
| Perplexity | 962 | 170 | 109 |

# Agenda



- Introduction to N-grams
- Estimating N-gram probabilities
- Evaluating language model
- Generalization and zeros
- Smoothing techniques:
  - Add-one (Laplace) smoothing
  - Interpolation, Backoff

# Zero probabilities



- Training set:
  ... denied the allegations
  ... denied the reports
  ... denied the claims
  ... denied the request

  P("offer" | denied the) = 0

- Test set
  ... denied the offer
  ... denied the loan

# Problems with Zero probabilities



- We underestimate the probability of all sorts of words that might occur
- The entire probability of the test set is 0.
  - So, we cannot calculate perplexity

# Agenda



- Introduction to N-grams
- Estimating N-gram probabilities
- Evaluating language model
- Generalization and zeros
- Smoothing techniques:
  - Add-one (Laplace) smoothing
  - Interpolation, Backoff

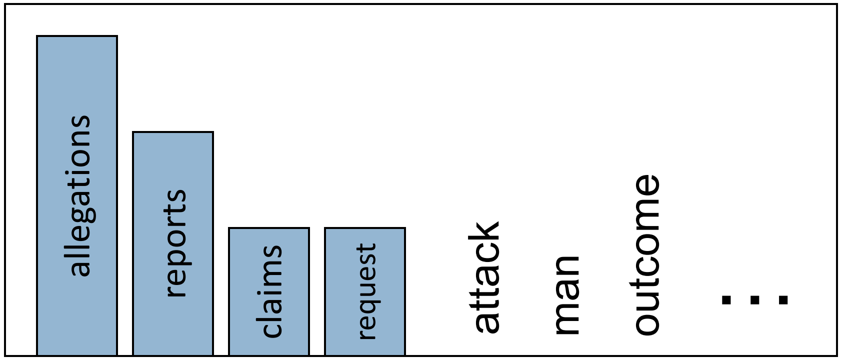
# Basic idea of smoothing (discounting) techniques



- When we have sparse statistics:

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

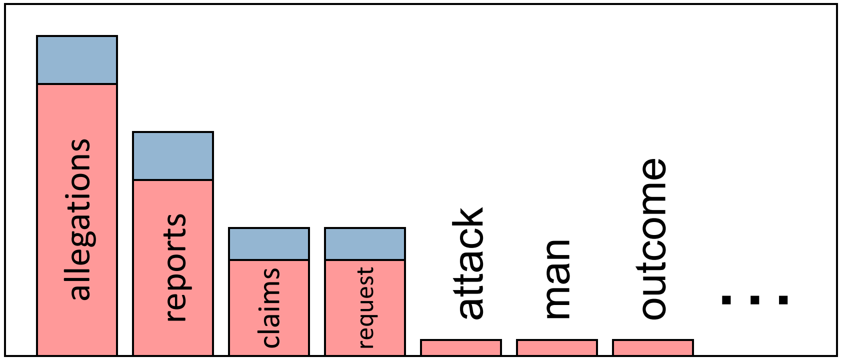
  7 total




- 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



# Laplace smoothing



- Add one to all the counts
- Pretend we saw each word one more time than we did

- MLE unigram probabilities:

$$P_{ML}(w_i) = \frac{c(w_i)}{N}$$

- Add-1 estimate:

$$P_{\mathrm{Laplace}} = \frac{c(w_i) + 1}{\sum_w (c(w) + 1)} = \frac{c(w_i) + 1}{N + V}$$

# Laplace smoothing: unigram model



i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

P(nara) = 1/20 = 0.05
P(i) = 2/20 = 0.1
P(</s>) = 3/20 = 0.15
P(kyoto) = 0/20 = 0

Vocab = {i, live, in, osaka, am, gradudate, student, my, school, is, nara, </s>}

V = 12

# Laplace smoothing: unigram model



i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

Vocab = {i, live, in, osaka, am, gradudate, student, my, school, is, nara, </s>}

V = 12

P(nara) = (1+1)/(20+12) = 0.0625
P(i) = (2+1)/(20+12) = 0.09375
P(</s>) = (3+1)/(20+12) = 0.125
P(kyoto) = (0+1)/(20+12) = 0.03125

# Laplace smoothing: bigrams



- MLE estimate:

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

- Add-1 estimate:

$$P_{\text{Laplace}} = \frac{c(w_{i-1}, w_i) + 1}{\sum_w (c(w_{i-1} w) + 1)} = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

# 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



$$P^*(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

| | 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

$$c^*(w_{i-1}w_i) = \frac{[c(w_{i-1}w_i) + 1] \times c(w_{i-1})}{c(w_{i-1}) + V}$$

| | 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 |

# 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 |

# Agenda



- Introduction to N-grams
- Estimating N-gram probabilities
- Evaluating language model
- Generalization and zeros
- Smoothing techniques:
  - Add-one (Laplace) smoothing
  - Interpolation, Backoff

# Linear interpolation



- Mix trigrams, bigrams and unigrams

  $$P(w_i|w_{i-2}, w_{i-1})$$
  $$= \lambda_1 \times P_{ML}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 \times P_{ML}(w_i|w_{i-1})$$
  $$+ \lambda_3 \times P_{ML}(w_i)$$

  where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all $i$

- Sometimes, all trigrams, bigrams, unigrams do not exist
  - Recall: $P(w_i) = \lambda \times P_{ML}(w_i) + (1 - \lambda) \times \frac{1}{N}$

# How to set 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:

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

# Example: bigrams

i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

- Maximum-likelihood estimation:
  - P(osaka | in) = c(in osaka)/c(in) = 1/2 = 0.5
  - P(nara | in) = c(in nara)/c(in) = 1/2 = 0.5
  - P(school | in) = c(in school)/c(in) = 0/2 = 0

# Example: interpolation



i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

- Using interpolation
  - $P(\text{school} \mid \text{in}) = \lambda_2 P_{ML}(\text{school} \mid \text{in}) + (1 - \lambda_2) P(\text{school})$
  - $P(\text{school}) = \lambda_1 P_{ML}(\text{school}) + (1 - \lambda_1)\frac{1}{N}$
  $$= \lambda_1 \times \frac{1}{20} + (1 - \lambda_1) \times \frac{1}{N}$$

# Unknown words: open vs. closed vocabulary



- 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

# Unknown words: open vs. closed vocabulary



- 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

# Advanced smoothing methods



- Kneser-Ney Smoothing
- Good-Turing Smoothing
- Katz's back-off