

# Language Modelling

## Class 03: Co-occurring words

- probability of word sequences
- predicting upcoming words
- dealing with unknown words in text



VRIJE  
UNIVERSITEIT  
BRUSSEL

6 March 2018

Katrien Beuls

Based on Jurafsky & Martin 2017, Ch.4

# Probabilistic Language Models

- Today's goal: assign a probability to a sentence
  - Machine Translation:
    - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
  - Spell Correction
    - The office is about fifteen **minuets** from my house
      - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
  - Speech Recognition
    - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
  - + Summarization, question-answering, etc., etc.!!

Why?

# Probabilistic Language Modeling

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

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

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

- Better: **the grammar**      But **language model** or **LM** is standard

# How to compute $P(W)$

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

## Reminder: The Chain Rule

- Recall the definition of conditional probabilities

$$p(B|A) = P(A,B)/P(A) \quad \text{Rewriting: } P(A,B) = P(A)P(B|A)$$

- More variables:

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

- The Chain Rule in General

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

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

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

$P(\text{"its water is so transparent"}) =$

$P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water})$

$\times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so})$

## How to estimate these probabilities

- Could we just count and divide?

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

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

# Markov Assumption

- Simplifying assumption:



Andrei Markov

$$P(\text{the lits water is so transparent that}) \approx P(\text{the l that})$$

- Or maybe

$$P(\text{the lits water is so transparent that}) \approx P(\text{the l transparent that})$$

# Markov Assumption

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

- In other words, we approximate each component in the product

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

# Simplest case: Unigram model

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

Some automatically generated sentences from a unigram model on WSJ

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:

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

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

## 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**:
- “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

# Estimating N-gram probabilities

# Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

# An example

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

< s > I am Sam < /s >

< s > Sam I am < /s >

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

$$P(I | < s >) = \frac{2}{3} = .67$$

$$P(Sam | < s >) = \frac{1}{3} = .33$$

$$P(am | I) = \frac{2}{3} = .67$$

$$P(< /s > | Sam) = \frac{1}{2} = 0.5$$

$$P(Sam | am) = \frac{1}{2} = .5$$

$$P(do | I) = \frac{1}{3} = .33$$

## 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

- 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

$P(< s > \text{ I want english food } </ s >) =$

$$P(I | < s >)$$

$$\times P(\text{want} | I)$$

$$\times P(\text{english} | \text{want})$$

$$\times P(\text{food} | \text{english})$$

$$\times P(</ s > | \text{food})$$

$$= .000031$$

## What kinds of knowledge?

- $P(\text{english} | \text{want}) = .0011$
- $P(\text{chinese} | \text{want}) = .0065$
- $P(\text{to} | \text{want}) = .66$
- $P(\text{eat} | \text{to}) = .28$
- $P(\text{food} | \text{to}) = 0$
- $P(\text{want} | \text{spend}) = 0$
- $P(\text{i} | \langle s \rangle) = .25$

# What kinds of knowledge?

- $P(\text{english} | \text{want}) = .0011$  cultural preferences
- $P(\text{chinese} | \text{want}) = .0065$
- $P(\text{to} | \text{want}) = .66$  grammatical regularities
- $P(\text{eat} | \text{to}) = .28$
- $P(\text{food} | \text{to}) = 0$
- $P(\text{want} | \text{spend}) = 0$
- $P(\text{i} | \langle s \rangle) = .25$  register characteristics

## Practical Issues

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

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

# 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 indispensable 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://ngrams.googlecode.com/>

# **Evaluation: Extrinsic versus Intrinsic Perplexity**

# 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?
- 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!
- And violates the honor code

## Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B is to put each model to use in a real-life 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 (in-vivo) 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.

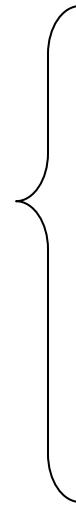
# Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and \_\_\_\_\_

The 33<sup>rd</sup> President of the US was \_\_\_\_\_

I saw a \_\_\_\_\_
  - 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



mushrooms 0.1  
pepperoni 0.1  
anchovies 0.01  
....  
fried rice 0.0001  
....  
and 1e-100

# Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest  $P(\text{sentence})$

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

Chain rule:

For bigrams:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

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

Minimizing perplexity is the same as maximizing probability

# Perplexity as branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign  $P=1/10$  to each digit?

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

## Lower perplexity = better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

# **The problem of generalisation and zero N-grams**

# The Shannon Visualization Method

- Choose a random bigram  $(\langle s \rangle, w)$  according to its probability
- Now choose a random bigram  $(w, x)$  according to its probability
- And so on until we choose  $\langle /s \rangle$
- Then string the words together

$\langle s \rangle$  I  
I want  
want to  
to eat  
eat Chinese  
Chinese food  
food  $\langle /s \rangle$

I want to eat Chinese food

# Approximating Shakespeare

1  
gram

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

## 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

# The wall street journal is not shakespeare (no offense)

1  
gram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

2  
gram

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

3  
gram

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

# Can you guess the author of these random 3-gram sentences?

- They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and gram Brazil on market conditions
- This shall forbid it should be branded, if renown made it empty.
- “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

# Can you guess the author of these random 3-gram sentences?

- They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and gram Brazil on market conditions **WSJ**
- This shall forbid it should be branded, if renown made it empty. **Shakespeare**
- “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for. **Jane Austin**

# The perils of overfitting

- 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

# Zeros

- Training set:
  - ... denied the allegations
  - ... denied the reports
  - ... denied the claims
  - ... denied the request
- Test set
  - ... denied the offer
  - ... denied the loan

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

## Zero probability bigrams

- 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)!

# **Smoothing: Laplace (add-one)**

# The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w | \text{denied the})$

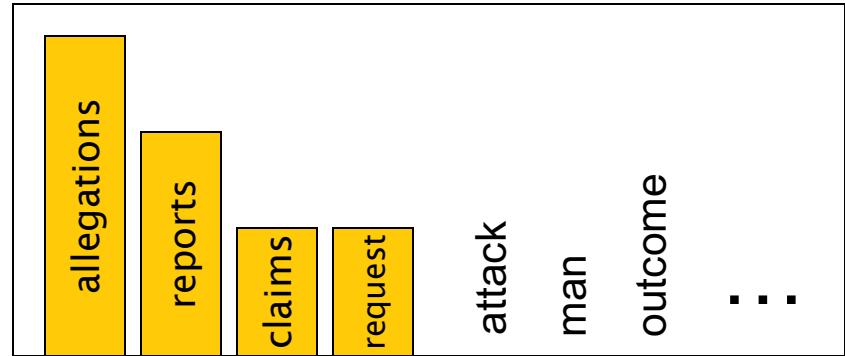
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

$P(w | \text{denied the})$

2.5 allegations

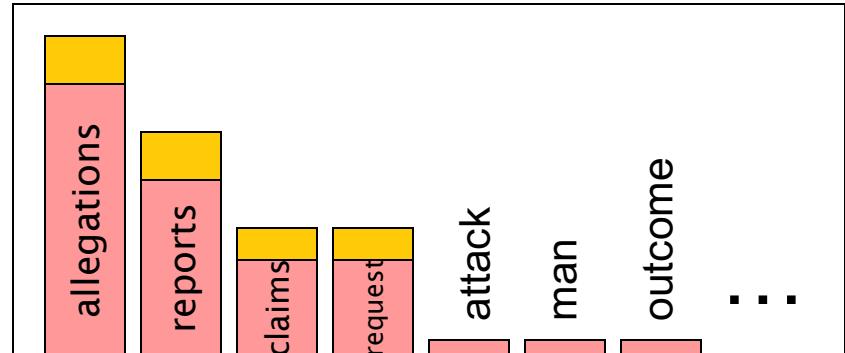
1.5 reports

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!

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

- MLE estimate:

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

- Add-1 estimate:

# 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	<b>0.00025</b>	0.0025	<b>0.00025</b>	<b>0.00025</b>	<b>0.00025</b>	0.00075
want	0.0013	<b>0.00042</b>	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	<b>0.00026</b>	0.0013	0.18	0.00078	<b>0.00026</b>	0.0018	0.055
eat	<b>0.00046</b>	0.00046	0.0014	<b>0.00046</b>	0.0078	0.0014	0.02	<b>0.00046</b>
chinese	0.0012	0.00062	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	0.052	0.0012	<b>0.00062</b>
food	0.0063	0.00039	0.0063	<b>0.00039</b>	0.00079	0.002	<b>0.00039</b>	0.00039
lunch	0.0017	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	0.0011	<b>0.00056</b>	<b>0.00056</b>
spend	0.0012	<b>0.00058</b>	0.0012	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>

# 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.

## **Smoothing:**

- **Interpolation versus Back-off**
- **Web-scale Language Models**

# 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  $\lambda$ s to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for  $\lambda$ s that give largest probability to held-out set:

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

# 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

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

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

# 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

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

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

## **Smoothing:**

- Kneser-Ney Smoothing**

# Absolute discounting: just subtract a little from each count

- Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros
- How much to subtract ?
- Church and Gale (1991)'s clever idea
- Divide up 22 million words of AP Newswire
  - Training and held-out set
  - for each bigram in the training set
  - see the actual count in the held-out set!

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

# Absolute Discounting Interpolation

- Save ourselves some time and just subtract 0.75 (or some d)!

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(w_i)$$

discounted bigram      Interpolation weight

unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)
- But should we really just use the regular unigram  $P(w)$ ?

# Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading \_\_\_\_\_?*  
“Francisco” is more common than “glasses”
  - ... but “Francisco” always follows “San”
- The unigram is useful exactly when we haven’t seen this bigram!
- Instead of  $P(w)$ : “How likely is  $w$ ”
- $P_{\text{continuation}}(w)$ : “How likely is  $w$  to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{\text{CONTINUATION}}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

# Kneser-Ney Smoothing II

- How many times does  $w$  appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalized by the total number of word bigram types

$$\left| \{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\} \right|$$

$$P_{CONTINUATION}(w) = \frac{\left| \{w_{i-1} : c(w_{i-1}, w) > 0\} \right|}{\left| \{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\} \right|}$$

# Kneser-Ney Smoothing III

- Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

# Kneser-Ney Smoothing IV

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

$\lambda$  is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

the normalized discount

The number of word types that can follow  $w_{i-1}$   
= # of word types we discounted  
= # of times we applied normalized discount

# Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} \text{count}(\bullet) & \text{for the highest order} \\ \text{continuation count}(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for  $\bullet$   
71

# **Summary & Assignment**

# Summary N-gram Language Models

- Language models can:
  - assign a **probability** to a **sequence** of words (e.g. a sentence)
  - **predict** a word from preceding words.
- N-gram LMs = **Markov models** estimating words from a fixed window of previous words, with probabilities estimated from normalized corpus frequencies (MLE)
- N-gram LMs can be evaluated **extrinsically** in a task or  
**intrinsically** using **perplexity**.
- Smoothing algorithms allow to estimate the probabilities of **unseen N-grams** using lower-order n-grams as **Back-off** or for **Interpolation**
- Kneser-Ney smoothing makes use of the probability of a word being **a novel continuation**.

# 1<sup>st</sup> Assignment

## SUMMARY

- Use language modelling to detect in which language variety a document is written.
- Build trigram letter language models for three varieties
- generate random output
- score test sentences with each LM
- determine the language variety based on perplexity
- report precision on the variety guessing task

# 1<sup>st</sup> Assignment

## TASKS

You will need to write a program in Python (3.x) that does the following (without using NTLK!):

- builds the Language model: reads in a text file, collects counts for all letter 3-grams, estimates probabilities, and writes out the model into a file
- generates random output strings of a given length  $k$  ( $3 < k < 300$ ), according to its probabilistic model
- reads in a test document, applies the language model to it, and outputs its perplexity

You may make the following simplifications:

- lowercase all characters
- remove all characters that are not ASCII alphabetic (i.e. digits, commas, periods, accented characters etc.)
- replace single whitespaces between words by a double underscore. (explain in your report why this is useful)
- State any additional assumptions and design decisions in your report

# 1<sup>st</sup> Assignment

## DATA

The data for this assignment will be available from 12 March 2018, 12pm, via  
<https://owncloud.vub.ac.be/index.php/s/PKEelJwHM7gQDgm>

The data consist of:

- training.gb – British English training data (scrambled sentences)
- training.us - American English training data (scrambled sentences)
- training.au – Australian English training data (scrambled sentences)
- test – labelled test sentences

# 1<sup>st</sup> Assignment

## REPORT

Your report should discuss the following issues:

- Do you need to run the test set on all three language models or is the score from a single variety model sufficient?
- Would a unigram or bigram language model work as well? Explain why (not).
- Do the language models show anything about similarity of the varieties? Why (not)?
- Can you think of a better way to make a language variety guesser?

# 1<sup>st</sup> Assignment

## REPORT

Include in the report:

- An excerpt of the language model for British English and another excerpt for American English model, displaying all n-grams and their probability with the two-letter history *i z* (E.g. *izo, ize, izo* etc.).
- 200 characters of random output for each of the three languages varieties
- the perplexity scores from the three language models for each test sentence
- precision across all test sentences
- the code of your programme(s)

NOTE: A report is a running text that also briefly sets the context and introduces the problem at stake. It further explains what you did and the results of your actions, followed by a discussion.

# 1<sup>st</sup> Assignment

## SUBMISSION

- **Friday 23 March 2018 at 12:00pm**
- as a **PDF file + code**
- Via PointCarré (preferred)
- via e-mail ([katrien@ai.vub.ac.be](mailto:katrien@ai.vub.ac.be))