

# Natural Language Processing

## Lecture 4: Language Modeling

Salima Lamsiyah

University Luxembourg, FSTM, DCS, MINE Research Group

[Salima.lamsiyah@uni.lu](mailto:Salima.lamsiyah@uni.lu)

# Lecture Plan

1. What is a language model?
2. Introduction to N-grams
3. Estimating N-grams probabilities
4. Evaluating Language Models
5. Generalization and zeros
6. Summary

# Beyond the Bag-of-Words!

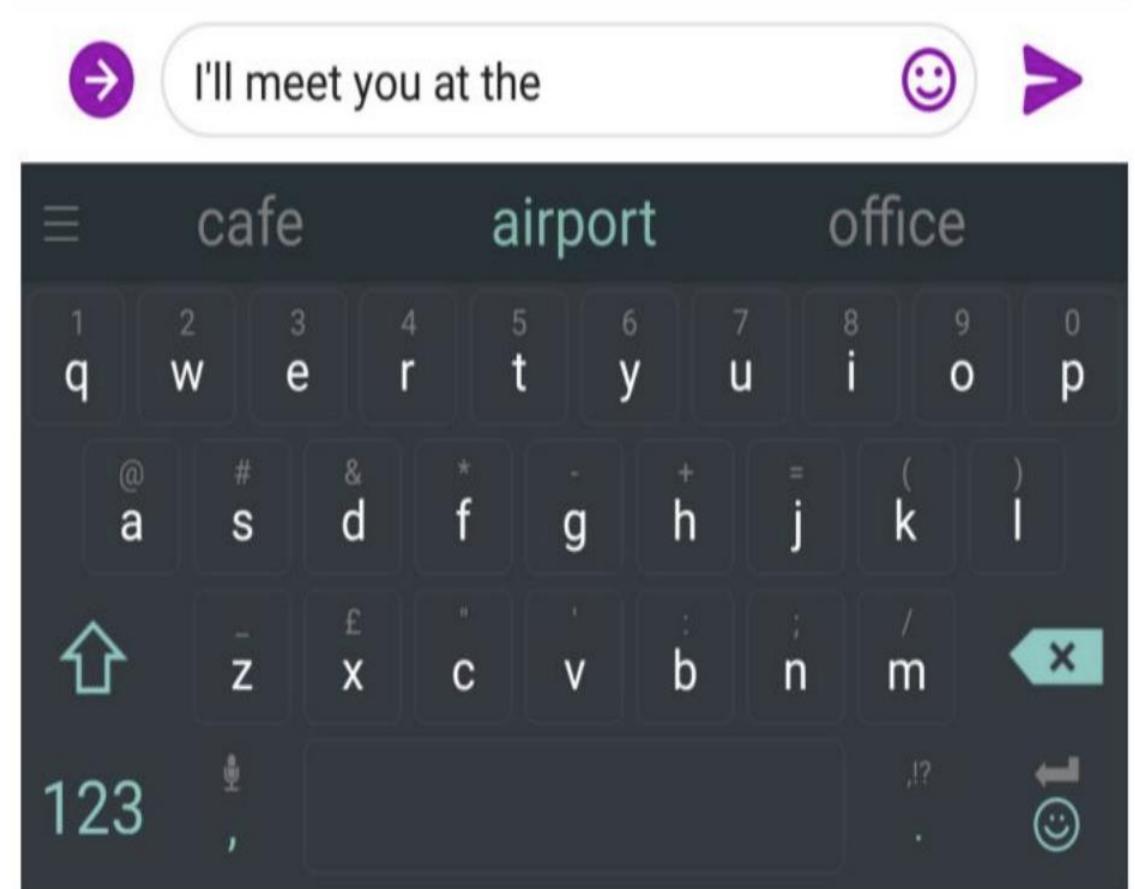
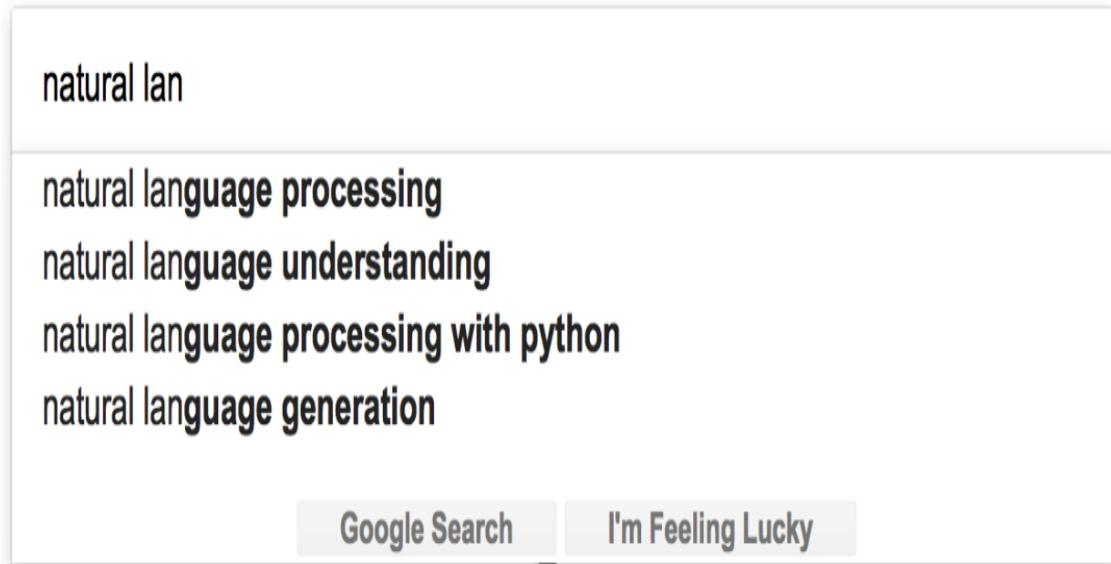
- Some linguistic phenomena require going beyond the bag-of-words:
  - *That's not bad for the first day*
  - *This is not the worst thing that can happen*
  - 
  - *It would be nice if you acted like you understood*
  - *This film should be brilliant. The actors are first grade. Stallone plays a happy, wonderful man. His sweet wife is beautiful and adores him. He has a fascinating gift for living life fully. It sounds like a great plot, however, the film is a failure.*

# Predicting words

The desert sky at night is filled with ...

- stars
  - light
  - silence
- \*Banana  
\*books

# Why word prediction?



# Why word prediction?

- Speech Recognition



*probability(I saw a van) > probability(eyes awe of an)*

- Machine Translation

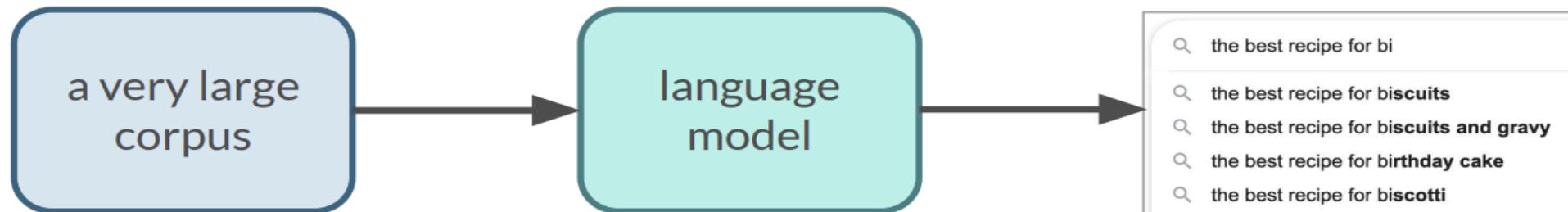
*probability(strong winds) > probability(large winds)*

# Why word prediction?

- **Spelling Correction**
  - “He entered the **ship** to buy some groceries”
  - $\text{probability}(\text{entered the shop to buy}) > \text{probability}(\text{entered the ship to buy})$
- **Augmentative Communication**
  - Predict most likely words from menu for people unable to physically talk or sign.

# Why word prediction?

- Create a Language Model (LM) from text corpus for **Word Prediction**



Autocomplete Sentence System

# What is A Language Model?

- A language model is a machine learning model that predicts upcoming words.
- An LM assigns a probability to each potential next word
  - = gives a probability distribution over possible next words
- An LM assigns a probability to a whole sentence
- It's how **large language models (LLMs)** work!
  - LLMs are **trained** to predict words
    - Left-to-right (autoregressive) LMs learn to predict next word
  - LLMs **generate** text by predicting words
    - By predicting the next word over and over again

# Language Modeling Methods

*Language  
Models  
Methods*

**Symbolic Rule-Based  
Methods (1950s-  
1980s)**

- Eliza (1964-66)
- SHRDLU (early 1970s)
- Conceptual Ontologies (MARGIE, SAM, PAM)
- ...

**Statistical-based  
Methods (1990s-  
2010s)**

- N-grams Models
- Hidden Markov Models
- ...

**Neural-based  
Methods  
(2010s- 2025)**

- Word2vec, GloVe
- RNN, LSTMS, GRU
- Transformer
- LLMS

# Statistical Language Models (SLMs) (This lecture)

- These models rely on statistical techniques and frequency-based approaches to predict words or sequences.
- ***Example:*** If we often say "bread and butter," it learns to predict "butter" after "bread and."
- ***Methods:*** n-gram models, backoff estimation, ...

# Neural Language Models (NLMs) (Next Lectures)

These models rely on deep learning techniques to capture complex patterns and long-term dependencies in language data.

- **Methods:**
- Artificial Neural Networks (*Word2vec, ...*)
- Recurrent Neural Networks (*ELMo, ULMFiT, ...*)
- Transformers (*BERT, Llama, GPT, ...*)
- ...

# Language Modeling (LM) more formally

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

$$\text{Probability}(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

- **Related Task:** predicting probability of an upcoming word:

$$\text{Probability}(w_n | w_1, w_2, w_3, \dots, w_{n-1})$$

- 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**.

- \*Note: let's call these **words** for now, but for LLMs we instead use **tokens**

# How to compute $P(W)$ ?

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

$$\text{Probability}(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

**Example:**  $P(\text{The cat sat on the mat})$

- **Related Task:** predicting probability of an upcoming word:

$$\text{Probability}(w_{t+1} | w_1, w_2, w_3, \dots, w_t)$$

**Example:**  $P(\text{mat} | \text{The cat sat on the})$

# How to estimate these probabilities $P(W)$ ?

- Could we just count and divide?

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\text{Count}(\text{The cat sat on the mat})}{\text{Count}(\text{The cat sat on the})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these
- Intuition: let's rely on **the Chain Rule of Probability**

# Reminder: The Chain Rule

- Recall the definition of conditional probabilities:
  - $p(B|A) = P(A, B)/P(A)$  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

$$P(w_1, w_2, w_3, \dots, w_n) = \prod_i P(w_i | w_1, w_2, \dots, w_{i-1})$$

$P(\text{The cat sat on the mat}) = P(\text{the}) \times P(\text{cat}|\text{the}) \times P(\text{sat}|\text{the cat}) \times P(\text{on}|\text{the cat sat}) \times P(\text{the}|\text{the cat sat on}) \times P(\text{mat}|\text{the cat sat on the})$



Implicit Order

# Markov Assumption: Simplifying assumption

- **Markov Assumption:** Use only the recent past to predict the next word

$P(\text{mat} | \text{The cat sat on the}) \approx P(\text{mat}|\text{the})$  (1<sup>st</sup> Order)

- Or maybe

$P(\text{mat} | \text{The cat sat on the}) \approx P(\text{mat}|\text{on the})$  (2<sup>nd</sup> Order)



Andrei Markov

# Markov Assumption More Formally!

- *Instead of:*

$$P(w_1, w_2, w_3, \dots, w_n) = \prod_i P(w_i | w_1, w_2, \dots, w_{i-1})$$

- *We use:*

$$P(w_1, w_2, w_3, \dots, w_n) \approx \prod_i P(w_i | w_{i-k}, \dots, w_{i-1})$$

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

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

# *N-grams Models*

---

# What is N-grams?

- $N$ -grams is a sequence of  $N$  words where the words order matter.
- $N$ -grams can be also characters or other elements.
- Example: **Corpus:** {I am happy because I am learning}
  - **Unigrams:** {I, am, happy, because, learning}
  - **Bigrams:** {I am, am happy, happy because, ...}
  - **Trigrams:** {I am happy, am happy because, ...}
  - **4-grams:** {I am happy because, am happy because I, ...}
  - ...

# N-grams Models: Examples

- Simplest case: Unigram model

$$P(w_1, w_2, w_3, \dots, w_n) \approx \prod_i P(wi)$$

- Bigram Model:

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

- Trigram Model:

$$P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(wi|w_{i-2}w_{i-1})$$

# Problems with N-gram models

1. N-grams can't handle **long-distance dependencies**:

*“She is from Morocco, so it makes sense that her first language is ...”*

- So Larger the N, more accurate and better the language model (**but also higher costs**)
2. N-grams don't do well at modeling new sequences (even if they have similar meanings to sequences they've seen)

The solution: **Large language models**

- can handle much longer contexts
- because they use neural **embedding spaces**, can model meaning better

# Why study N-gram models?

- They are the "fruit fly of NLP", a simple model that introduces important topics for **large language models**
  - **training** and **test** sets
  - the **perplexity** metric
  - **sampling** to generate sentences
  - ideas like **interpolation** and **backoff**
- N-grams are efficient and fast and useful in situations where LLMs are too heavyweight

# *Estimating N-grams 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 >

$$P(\text{I} | \text{<s>}) =$$

$$P(\text{</s>} | \text{Sam}) =$$

$$P(\text{Sam} | \text{<s>}) =$$

$$P(\text{Sam} | \text{am}) =$$

$$P(\text{am} | \text{I}) =$$

$$P(\text{do} | \text{I}) =$$

# 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 >

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

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

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

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

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

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

# Example: Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- tell me about chez panisse
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

# Raw bigram counts

## Berkeley Restaurant Project sentences

- 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

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

- 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

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

$P(\text{I} | < s >)$

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

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

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

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

$= .000031$

# What kinds of knowledge do N-grams represent?

- $P(\text{to} \mid \text{want}) = .66$
- $P(\text{want} \mid \text{spend}) = 0$
- $P(\text{of} \mid \text{to}) = 0$

- $P(\text{dinner} \mid \text{lunch or}) = .83$
- $P(\text{dinner} \mid \text{for}) \sim P(\text{lunch} \mid \text{for})$

- $P(\text{chinese} \mid \text{want}) > P(\text{english} \mid \text{want})$

## Knowledge of grammar

("want" is followed by infinitive "to")  
("of" is not a verb)

## Knowledge of meaning

The words "dinner" or "lunch" are semantically related.

## Knowledge about the world

Chinese food is very popular

# Dealing with underflow!

- LM probabilities are stored and computed in log format, i.e. **log probabilities**
- This avoids underflow from multiplying many small numbers

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

- If we need probabilities we can do one exp at the end

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

# Larger N-grams

- 4-grams, 5-grams
- Large datasets of large n-grams have been released
  - N-grams from Corpus of Contemporary American English (COCA) 1 billion words (Davies 2020)
  - Google Web 5-grams (Franz and Brants 2006) 1 trillion words)
  - Efficiency: quantize probabilities to 4-8 bits instead of 8-byte float

Newest model: infini-grams ( $\infty$ -grams) (Liu et al 2024)

- No precomputing! Instead, store 5 trillion words of web text in **suffix arrays**. Can compute n-gram probabilities with any n!

# N-gram LM Toolkits

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

# *Evaluating language models*

---

# How good is our model?

- Does our model assign higher probability to typical, grammatically correct sentences?
- There are two ways to evaluate the N-grams models:
  - **Extrinsic** and
  - **Intrinsic** Evaluation metrics

# Extrinsic (in-vivo) evaluation of N-gram models

- Best evaluation for comparing models A and B
  1. Put each model in a task
    - spelling corrector, speech recognizer, Machine Translation system
  2. Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  3. Compare accuracy for A and B
- **Extrinsic evaluation are time-consuming; can take days or weeks**
- **Doesn't always generalize to other applications**

## Intrinsic (in-vitro) Evaluation: Perplexity (PP)

- Directly measures language model performance at predicting words.
- Doesn't necessarily correspond with real application performance
- But gives us a single general metric for language models
- Useful for large language models (LLMs) as well as n-grams

# Training sets and test sets

- 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** (like **perplexity**) tells us how well our model does on the test set.

# Choosing training and test sets

- If we're building an LM for a specific task
  - The test set should reflect the task language we want to use the model for
- If we're building a general-purpose model
  - We'll need lots of different kinds of training data
  - We don't want the training set or the test set to be just from one domain or author or language.

# Training on the test set

- We can't allow test sentences into the training set
  - Or else the LM will assign that sentence an artificially high probability when we see it in the test set
  - And hence assign the whole test set a falsely high probability.
  - Making the LM look better than it really is
- This is called “**Training on the test set**”
- Bad science!

# Dev sets

- If we test on the test set many times we might implicitly tune to its characteristics
  - Noticing which changes make the model better.
- So we run on the test set only once, or a few times
- That means we need a third dataset:
  - A **development test set** or, **devset**.
  - We test our LM on the devset until the very end
  - And then test our LM on the **test set** once

# Intuition of perplexity as evaluation metric: How good is our language model?

- Intuition: A good LM prefers "real" sentences
- Assign higher probability to “real” or “frequently observed” sentences
- Assigns lower probability to “word salad” or “rarely observed” sentences?

# Intrinsic (in-vitro) Evaluation: Perplexity (PP)

- The best language model is the one that predicts unseen words or sentences most accurately by giving them the highest probability  $P(W)$ .
- Perplexity is the inverse probability of the test set, normalized by the number of words:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

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

For bigrams:

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

**Minimizing perplexity is the same as  
maximizing probability**

# Practical Issues: *Log Perplexity*

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

$$\begin{aligned} PP(W) &= \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1, \dots, w_{i-1})}} \\ &= \frac{-1}{N} \sum_{i=1}^N \log P(w_i | w_1, \dots, w_{i-1}) \end{aligned}$$

# Examples

Training 38 million words, test 1.5 million words, WSJ corpus  
Perplexity Unigram: 962 Bigram: 170 Trigram: 109

## Unigram

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

## Bigram

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

## Trigram

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

[Figure from *Speech and Language Processing* by Dan Jurafsky et. al]

# Lower perplexity = better model

- Training 38 million words, test 1.5 million words, Wall Street Journal corpus

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

- Good language models have perplexity scores between 60 to 20 sometimes even lower for English.

# *Generalizations and zeros*

---

# Generalization of N-grams

- Generalization of N-grams
  - Not all N-grams will be observed in training data
  - Test corpus might have some that have zero probability
- Example
  - Training set: Google news
  - Test set: Shakespeare
  - $P(\text{affray} \mid \text{voice doth us}) = 0$    Therefore    $P(\text{test corpus}) = 0$
- **Undefined perplexity**

# Generalization of N-grams

- Training Set:

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

- Test Set:

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

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

# 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: **Dealing with Zeros!**
    - Things that don't ever occur in the training set
    - But occur in the test set

# *Smoothing: Add-one (Laplace) smoothing*

---

# 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 is:

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

- Add-1 estimate:

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

V is the size vocabulary

# Example: Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- tell me about chez panisse
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

# Raw bigram counts

## Berkeley Restaurant Project sentences

- 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

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

- 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

# 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>	<b>0.00046</b>	0.0014	<b>0.00046</b>	0.0078	0.0014	0.02	<b>0.00046</b>
chinese	0.0012	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	0.052	0.0012	<b>0.00062</b>
food	0.0063	<b>0.00039</b>	0.0063	<b>0.00039</b>	0.00079	0.002	<b>0.00039</b>	<b>0.00039</b>
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

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	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

Reconstituted counts

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

	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 (e.g. Interpolation and Backoff )
- But add-1 is used to smooth other NLP models
  - Or we can use **add-k** where  $0 < k < 1$
  - It's used in naive bayes text classification
  - In domains where the number of zeros isn't so huge.

# *Interpolation and Backoff*

---

# Backoff and Interpolation

- Sometimes it helps to use **less** context
    - Condition on less context for contexts you haven't learned much about
1. **Backoff:**
    - use trigram if you have good evidence,
    - otherwise bigram, otherwise unigram
  2. **Interpolation:**
    - mix unigram, bigram, trigram
    - **Interpolation works better**

# Linear Interpolation

- Simple interpolation

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

- Use a combination of models (trigrams, bigrams, unigram) to estimate probability
- Strong empirical performance

# How to set $\lambda$ s for interpolation?

- Use a **held-out (Dev set, Validation set)** corpus



- Choose  $\lambda$ s to maximize 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

# Backoff

Suppose you want:

$$P(\text{pancakes} \mid \text{delicious soufflé})$$

- If the trigram probability is 0, use the bigram
  - $P(\text{pancakes} \mid \text{soufflé})$
- If the bigram probability is 0, use the unigram
  - $P(\text{pancakes})$

# Summary: What to do if you never saw an n-gram in training

- **Smoothing:** Pretend you saw every n-gram one (or k) times more than you did
  - A blunt instrument (replacing a lot of zeros) but sometimes useful
- **Backoff:** If you haven't seen the trigram, use the (weighted) bigram probability instead
  - Weighting is messy; "stupid" backoff works fine at web-scale
- **Interpolation:** (weighted) mix of trigram, bigram, unigram
  - Usually the best! We also use interpolation to combine multiple LLMs

# *Advanced Method: Kneser-Ney Smoothing* Chapter 3

---

# Summary

- Language models offer a way to assign a probability to a sentence or other sequence of words, and to predict a word from preceding words.
- N-grams are Markov models that estimate words from a fixed window of previous words.
- N-gram probabilities can be estimated by counting in a corpus and normalizing (**the maximum likelihood estimate**).
- N-gram language models are evaluated **extrinsically** in some task, or **intrinsically** using **perplexity**

# Summary

- The perplexity of a test set according to a language model is the geometric mean of the inverse test set probability computed by the model.
- **Smoothing** algorithms provide a more sophisticated way to estimate the probability of N-grams.
- Commonly used smoothing algorithms for N-grams rely on lower-order N-gram counts through **backoff** or **interpolation**.
- Both **backoff** and **interpolation** require **discounting** to create a probability distribution.

# Reading

- Please refer to this chapter: [Chapter 3](#)
- <https://machinelearningmastery.com/what-is-maximum-likelihood-estimation-in-machine-learning/>

# Exercise

- <https://drive.google.com/file/d/17aLEhSroQShzMchs79qMhYcBXxmg0bop/view?usp=sharing>

## Next Class

- **Lecture 5: Text Classification**

