CS5984: Natural Language Processing

# Language Modeling

Lifu Huang

lifuh@vt.edu

Torgersen Hall, Suite 3160E

N L P L a b | Natural Language Processing Lab

## Language Modeling

- We have some (finite) vocabulary,
  - e.g., V = {the, a, man, telescope, saw, Beckham, two, fun, …}

- We have an (infinite) set of strings, V*
  - the STOP
  - a STOP
  - the fan STOP
  - the fan saw STOP
  - the fan saw saw STOP
  - the fan saw Beckham STOP
  - the fan saw Beckham play for Real Madrid STOP
  - ….  ….

# Language Modeling

- A language model over a vocabulary V assigns probabilities to string drawn from V*
- Probability: $P(w_1, w_2, \dots, w_n)$
  - P (the STOP) = $10^{-11}$
  - P (a STOP) = $10^{-12}$
  - P (the fan STOP) = $10^{-9}$
  - P (the fan saw STOP) = $10^{-8}$
  - P (the fan saw saw STOP) = $10^{-14}$
  - P (the fan saw Beckham STOP) = $10^{-7}$
  - P (the fan saw Beckham play for Real Madrid STOP) = $10^{-9}$
  - … …

## A Naïve Method

- We have *N* training sentences

- For any sentence $x_1, \dots, x_n$, $c(x_1, \dots, x_n)$ is the number of times the sentence is seen in the training data

- A naïve estimate:

$$p(x_1 \dots x_n) = \frac{c(x_1 \dots x_n)}{N}$$

- Problem: too sparse!

## Markov Process

- Assume given a sequence with length n

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

- First-Order Markov Processes
  - Each future variable (event) is only dependent on the current variable (event) and independent of past variables (events)

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) \\ = \;& P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \\ = \;& P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1}) \end{aligned}$$

## Second-Order Markov Processes

- Each variable (event) only depends on the two variables (events) before it

$$\begin{aligned}
& P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) \\
= \; & P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \\
& \times \prod_{i=3}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \\
= \; & \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})
\end{aligned}$$

- For convenience we assume $x_0 = x_{-1} = *$, where * is a special “START” symbol

# N-gram Language Models

- Markov Assumptions: an n-gram language model assumes each word depends only on the last n−1 words

$$P_{ngram}(w_1 ... w_i) \quad := \quad P(w_1)P(w_2|w_1)...P(\underbrace{w_i}_{nth\ word} \quad |\underbrace{w_{i-n-1}...w_{i-1}}_{prev.\ n-1\ words})$$

$$P(w_i|w_1...w_{i-1}) :\approx P(w_i|w_{i-n-1}...w_{i-1})$$

- Unigram model $P(w_1)P(w_2)...P(w_i)$
- Bigram model $P(w_1)P(w_2|w_1)...P(w_i|w_{i-1})$
- Trigram model $P(w_1)P(w_2|w_1)...P(w_i|w_{i-2}w_{i-1})$
- N-gram model $P(w_1)P(w_2|w_1)...P(w_i|w_{i-n-1}...w_{i-1})$

# Estimating N-gram Model

- 1. Bracket each sentence by special start and end symbols

  START $w_1$ $w_2$ $w_3$ $w_4$ $w_5$ $w_6$ STOP

- 2. Count the frequency of each n-gram

  e.g., Count ($w_1$ $w_2$) = 100

- 3. normalize these frequencies to get the probability – relative frequency estimate or Maximum Likelihood Estimation (MLE)

$$P(w_n|w_{n-1}) = \frac{count(w_{n-1}w_n)}{count(w_{n-1})}$$

## Example – Trigram Model

- For the sentence

  START the dog barks STOP

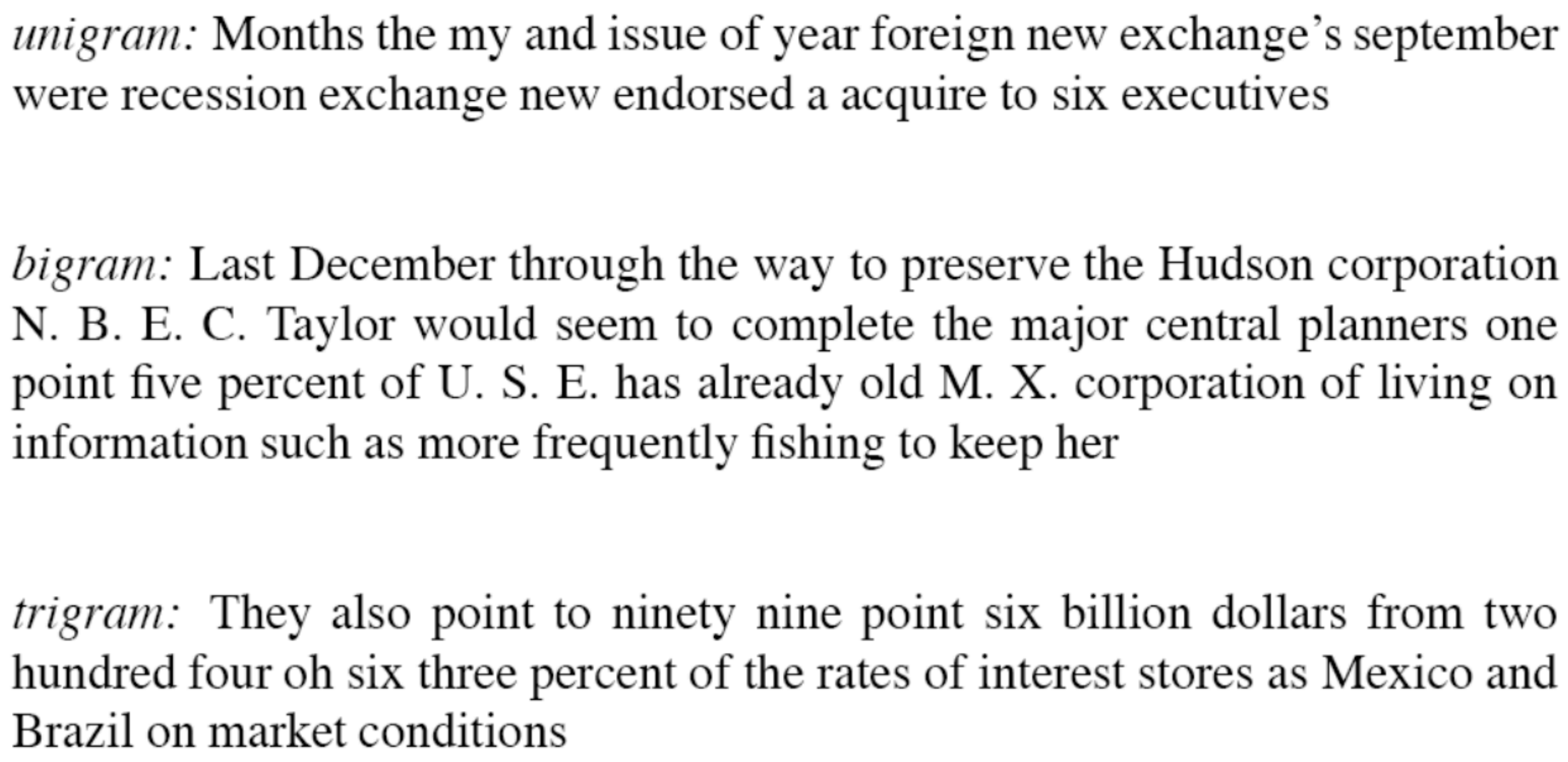
- Count(START the dog), Count(the dog barks), Count(dog barks STOP)
- $P(START\ the\ dog\ barks\ STOP) = p(the|START)$

$$p(dog|START, the)$$
$$p(barks|the, dog)$$
$$p(STOP|dog, barks)$$

$$p(barks|the, dog) = \frac{count(the\ dog\ barks)}{count(the\ dog)}$$

- Say our vocabulary size is N = | V |, then there are $N^3$ parameters in the model – Sparse!!

# Generating text with N-gram Language Model



*unigram:* Months the my and issue of year foreign new exchange's september were recession exchange 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

## Intrinsic vs Extrinsic Evaluation

- How do we know whether one language model is better than another?

- There are two ways to evaluate models
  - intrinsic evaluation captures how well the model captures what it is supposed to capture (e.g., probabilities) – independent of other applications
  - extrinsic (task-based) evaluation captures how useful the model is in a particular task

- Both cases require an evaluation metric that allows us to measure and compare the performance of different models

# Intrinsic Evaluation of Language Models - Perplexity

- Perplexity: the inverse of the probability of the test set (as assigned by the language model), normalized by the number of word tokens in the test set.

- Minimizing perplexity = maximizing probability!

- Language model $LM_1$ is better than $LM_2$ if $LM_1$ assigns lower perplexity (= higher probability) to the test corpus $w_1 \ldots w_N$
  - the perplexity of $LM_1$ and $LM_2$ can only be directly compared if both models use the same vocabulary

## Intrinsic Evaluation of Language Models - Perplexity

- Assume the test corpus has N tokens, $w_1 \dots w_N$ If the LM assigns probability $P(w_1, \dots, w_{i-n})$ to the test corpus, its perplexity, $PP(w_1 \dots w_N)$, is defined as

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

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

$$PP(w_1 \dots w_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}, \dots, w_{i-n+1})}}$$ (N-gram model)

- A LM with lower perplexity is better because it assigns a higher probability to the unseen test corpus

## Practical Issues

- Language model probabilities are usually very small, so multiplying them together often yields to underflow

- Replace the original probabilities with logarithms

$$PP(w_1 \dots w_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1}, \dots, w_{i-n+1})}}$$ (N-gram model)

$$\Big\downarrow$$

$$PP(w_1 \dots w_N) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i | w_{i-1}, \dots, w_{i-n+1}\right)$$

## Perplexity and LM Order

- Train Unigram, Bigram, Trigram LMs on the *Wall Street Journal* corpus (trained on 38M tokens, tested on 1.5M tokens, vocabulary: 20K word types)

| | **Unigram** | **Bigram** | **Trigram** |
|---|---|---|---|
| **Perplexity** | 962 | 170 | 109 |

- the more information the n-gram gives us about the word sequence, the lower the perplexity
  - Bigram LMs have lower perplexity than unigram LMs
  - Trigram LMs have lower perplexity than bigram LMs
  - …

$$PP(w_1 \ldots w_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1}, \ldots, w_{i-n+1})}}$$ (N-gram model)

## Extrinsic (Task-Based) Evaluation of LMs

- Perplexity tells us which LM assigns a higher probability to unseen text
- This doesn’t necessarily tell us which LM is better for our task (i.e. is better at scoring candidate sentences)
- Task-based evaluation:
  - Train model A, plug it into your system for performing task T
  - Evaluate performance of system A on task T.
  - Train model B, plug it in, evaluate system B on same task T.
  - Compare scores of system A and system B on task T.

## Word Error Rate (WER)

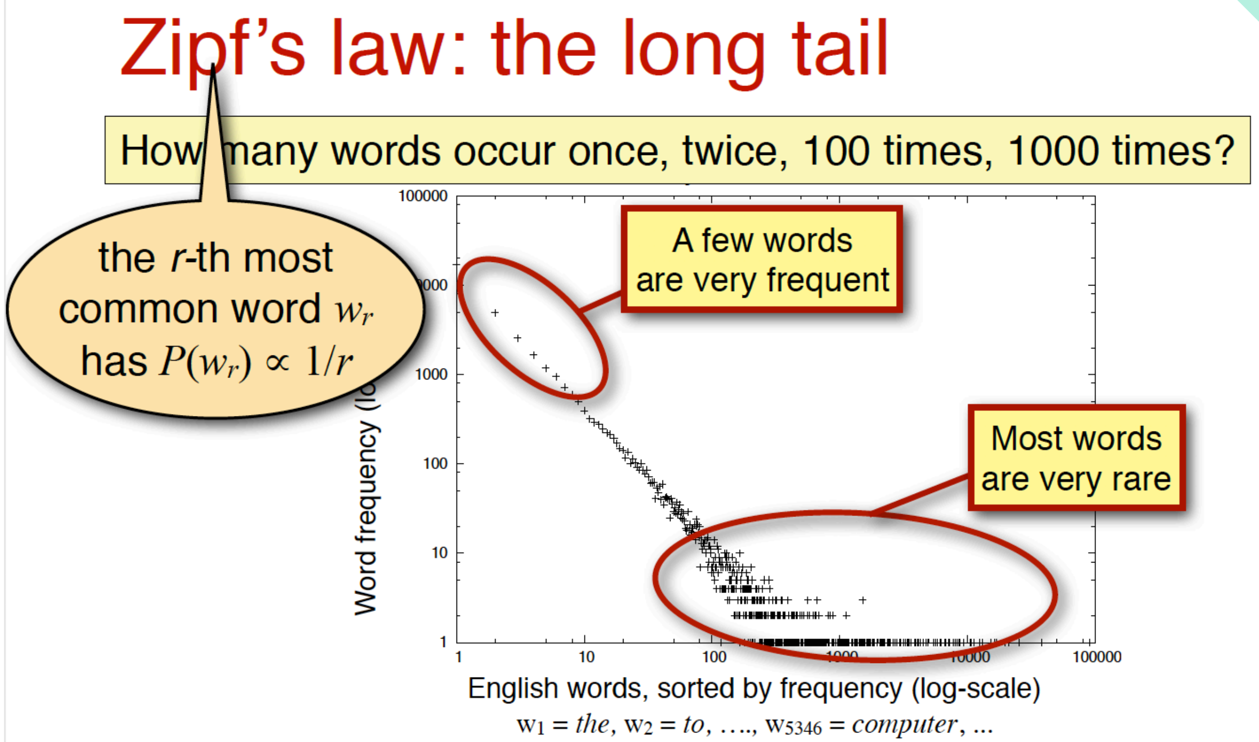
- Originally developed for speech recognition.
- How much does the predicted sequence of words differ from the actual sequence of words in the correct transcript?

$$\text{WER} = \frac{\text{Insertions} + \text{Deletions} + \text{Substitutions}}{\text{Actual words in transcript}}$$

- Insertions: “eat lunch” → “eat a lunch”
- Deletions: “see a movie” → “see movie”
- Substitutions: “drink ice tea”→ “drink nice tea”

## Sparse Data Problems

- Given the *Wall Street Journal* corpus, the vocabulary size is N=|V|=20K, there will be 400M Bigrams in total, while more than 99.9% of the possible bigrams don't occur in the corpus
- Zero probability for unseen words??
- We need language models to assign some probability to the unseen words and n-grams



In natural language:
- A small number of events (e.g. words) occur with high frequency
- A large number of events occur with very low frequency

# Laplace Smoothing

- Assume every (seen or unseen) event occurred once (or K-times) more than it did in the training dataset
- Add-One Laplace Smoothing Example: unigram probabilities

$$P(w_i) = \frac{C(w_i)}{\sum_j C(w_j)} = \frac{C(w_i)}{N} \quad \longrightarrow \quad P(w_i) = \frac{C(w_i)+\mathbf{1}}{\sum_j (C(w_j)+\mathbf{1})} = \frac{C(w_i)+\mathbf{1}}{N+\mathbf{V}}$$

- 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 |

$\longrightarrow$

| | 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 |

# Linear Interpolation

- We don’t see “Bob was reading”, but we see “__ was reading”.
- We estimate P(*reading* |’*Bob was*’) = 0 but P(*reading* | ‘*was*’) > 0

- Use (n –1)-gram probabilities to smooth n-gram probabilities:

$$
\begin{aligned}
\tilde{P}(w_i|w_{i-1},w_{i-2}) = & \lambda_3 \cdot \hat{P}(w_i|w_{i-1},w_{i-2}) \\
& + \lambda_2 \cdot \hat{P}(w_i|w_{i-1}) \\
& + \lambda_1 \cdot \hat{P}(w_i) \\
& \text{for } \lambda_1 + \lambda_2 + \lambda_3 = 1
\end{aligned}
$$

$$
\begin{aligned}
& P_{\text{smoothed}}(\mathrm{w_i} = \mathit{reading} \mid \mathrm{w_{i-1}} = \mathit{was}, \mathrm{w_{i-2}} = \mathit{Bob}) = \\
& \lambda_3\, P_{\text{unsmoothed-trigram}}(\mathrm{w_i} = \mathit{reading} \mid \mathrm{w_{i-1}} = \mathit{was}, \mathrm{w_{i-2}} = \mathit{Bob}) \\
& + \lambda_2\, P_{\text{unsmoothed-bigram}}(\mathrm{w_i} = \mathit{reading} \mid \mathrm{w_{i-1}} = \mathit{was}) \\
& + \lambda_1\, P_{\text{unsmoothed-unigram}}(\mathrm{w_i} = \mathit{reading})
\end{aligned}
$$

## How to estimate the $\lambda$ values?

- Method 1: Held-out Estimation
  - Divide data into training and held-out data
  - Estimate models on training data and use held-out data to find the $\lambda$ that gives the best model performance

- Method 2: allow $\lambda$ to vary, e.g., define a deterministic function of the frequencies of $w_{i-n} \ldots w_{i-1}$

$$
\begin{aligned}
q(w_i \mid w_{i-2}, w_{i-1}) = \quad & \lambda_1^{\Pi(w_{i-2},w_{i-1})} \times q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) \\
& + \lambda_2^{\Pi(w_{i-2},w_{i-1})} \times q_{\text{ML}}(w_i \mid w_{i-1}) \\
& + \lambda_3^{\Pi(w_{i-2},w_{i-1})} \times q_{\text{ML}}(w_i)
\end{aligned}
$$

where $\lambda_1^{\Pi(w_{i-2},w_{i-1})} + \lambda_2^{\Pi(w_{i-2},w_{i-1})} + \lambda_3^{\Pi(w_{i-2},w_{i-1})} = 1$,
and $\lambda_i^{\Pi(w_{i-2},w_{i-1})} \geq 0$ for all $i$.

$$
\Pi(w_{i-2}, w_{i-1}) = \begin{cases} 1 & \text{If Count}(w_{i-1}, w_{i-2}) = 0 \\ 2 & \text{If } 1 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 2 \\ 3 & \text{If } 3 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 5 \\ 4 & \text{Otherwise} \end{cases}
$$