



**NUS**  
National University  
of Singapore

| **Computing**

# **CS4248: Natural Language Processing**

## Lecture 3 — n-Gram Language Models

# Outline

- **Language Models**
  - **Motivation**
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- **Evaluating Language Models**

# Language Models — Motivation

- Which sentence makes more sense?  $S_1$  or  $S_2$ ?

Example 1:

$S_1$ : "on guys all I of noticed sidewalk three a sudden standing the"

$S_2$ : "all of a sudden I noticed three guys standing on the sidewalk"

Example 2:

$S_1$ : "the role was played by an ~~aeress~~across famous for her comedic timing"

$S_2$ : "the role was played by an ~~aeress~~actress famous for her comedic timing"

- But why?

- Probability of  $S_2$  higher than of  $S_1$ :  $P(S_2) > P(S_1)$

→ **Language Models** — Assigning probabilities to a sentence, phrase (or word)

# Language Models — Basic Idea

- 2 basic notions of probabilities

## (1) Probability of a sequence of words $W$

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

**Example:**  $P(\text{"remember to submit your assignment"})$

## (2) Probability of an upcoming word $w_n$

$$P(w_n \mid w_1, w_2, w_3, \dots, w_{n-1})$$

**Example:**  $P(\text{"assignment"} \mid \text{"remember to submit your"})$

In this lecture: How to calculate these probabilities?

# Language Models — Applications

- Language Models fundamental for many NLP task

- **Speech Recognition**  $P(\text{"we built this city on rock and roll"}) > P(\text{"we built this city on sausage rolls"})$

- **Spelling correction**  $P(\text{"... has no mistakes"}) > P(\text{"... has no mistaek"})$

- **Grammar correction**  $P(\text{"... has improved"}) > P(\text{"... has improve"})$

- **Machine Translation**  $P(\text{"I went home"}) > P(\text{"I went to home"})$

# Outline

- **Language Models**
  - Motivation
  - **Sentence Probabilities**
  - Markov Assumption
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- **Evaluating Language Models**

# Probabilities of Sentences (more generally: sequence of words)

$$\left. \begin{array}{l} P(\text{"remember to submit your assignment"}) \\ P(\text{"assignment"} \mid \text{"remember to submit your"}) \end{array} \right\} \rightarrow \text{How to calculate those probabilities?}$$

- Quick review: Chain Rule (allows the iterative calculation of joint probabilities)

- Chain rule for 2 random events:

$$P(A_1, A_2) = P(A_2|A_1) \cdot P(A_1)$$

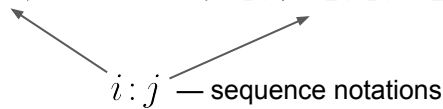
- Chain rule for 3 random events:

$$\begin{aligned} P(A_1, A_2, A_3) &= P(A_3|A_1, A_2) \cdot P(A_1, A_2) \\ &= P(A_3|A_1, A_2) \cdot P(A_2|A_1) \cdot P(A_1) \end{aligned}$$

- ...

# Probabilities of Sentences

- Chain rule — generalization to  $N$  random events

$$\begin{aligned} P(A_1, \dots, A_N) &= P(A_1) \cdot P(A_2|A_1) \cdot P(A_3|A_1: 2) \cdot \dots \cdot P(A_N|A_1: N-1) \\ &= \prod_{i=1}^N P(A_i|A_1: i-1) \end{aligned}$$


$i:j$  — sequence notations

→ Chain rule applied to sequences of words

$$\begin{aligned} P(w_1, \dots, w_N) &= P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1: 2) \cdot \dots \cdot P(w_N|w_1: N-1) \\ &= \prod_{i=1}^N P(w_i|w_1: i-1) \end{aligned}$$



# Quick Quiz



# Probabilities of Sentences

- Calculating the probabilities using Maximum Likelihood Estimations

$$P(w_n|w_{1:n-1}) = \frac{\text{Count}(w_{1:n-1}w_n)}{\sum_w \text{Count}(w_{1:n-1}w)} = \frac{\text{Count}(w_{1:n})}{\text{Count}(w_{1:n-1})}$$



# Probabilities of Sentences — Example

## (1) Application of Chain Rule

$$\begin{aligned}P(\text{"remember to submit your assignment"}) &= P(\text{"remember"}) \cdot \\&\quad P(\text{"to"} \mid \text{"remember"}) \cdot \\&\quad P(\text{"submit"} \mid \text{"remember to"}) \cdot \\&\quad P(\text{"your"} \mid \text{"remember to submit"}) \cdot \\&\quad P(\text{"assignment"} \mid \text{"remember to submit your"})\end{aligned}$$

## (2) Maximum Likelihood Estimation

$$P(\text{"remember"}) = \frac{\text{Count}(\text{"remember"})}{N}$$

$$P(\text{"to"} \mid \text{"remember"}) = \frac{\text{Count}(\text{"remember to"})}{\text{Count}(\text{"remember"})}$$

...

$$P(\text{"assignment"} \mid \text{"remember to submit your"}) = \frac{\text{Count}(\text{"remember to submit your assignment"})}{\text{Count}(\text{"remember to submit your"})}$$

Do you see any problems?

# Probabilities of Sentences — Problems

$$P(\text{"assignment"} \mid \text{"remember to submit your"}) = \frac{\text{Count}(\text{"remember to submit your assignment"})}{\text{Count}(\text{"remember to submit your"})}$$

- Problem: (very) long sequences

- Large number of entries in table with joint probabilities

- A sequence (or subsequence)  $w_{i:j}$  may not be present in corpus

$$\left. \begin{array}{l} \text{A sequence (or subsequence) } w_{i:j} \\ \text{may not be present in corpus} \end{array} \right\} \rightarrow \text{Count}(w_{i:j}) = 0 \rightarrow \prod_{n=1}^N P(w_n | w_{1:n-1}) = 0$$

(we can ignore  $\frac{0}{0}$  here; this can be handled in the implementation)

→ Can we keep the sequences short?

# Outline

- **Language Models**
  - Motivation
  - Sentence Probabilities
  - **Markov Assumption**
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- **Evaluating Language Models**

# Markov Assumption

- Probabilities depend on only on the last  $k$  words

$$P(w_1, \dots, w_N) = \prod_{n=1}^N P(w_n | w_{1:n-1}) = \prod_{n=1}^N P(w_n | w_{n-k:n-1})$$

- For our example:

$$P(\text{"assignment"} \mid \text{"remember to submit your"}) \approx P(\text{"assignment"} \mid \text{"your"})$$

$$P(\text{"assignment"} \mid \text{"submit your"})$$

$$P(\text{"assignment"} \mid \text{"to submit your"})$$

...

# n-Gram Models (consider the only $n-1$ last words)

**Unigram (1-gram):**  $P(w_n | w_1 : n-1) \approx P(w_n)$

**Bigram (2-gram):**  $P(w_n | w_1 : n-1) \approx P(w_n | w_{n-1})$

**Trigram (3-gram):**  $P(w_n | w_1 : n-1) \approx P(w_n | w_{n-2}, w_{n-1})$

# n-Gram Models

## Maximum Likelihood Estimation

**Unigram (1-gram):**  $P(w_n|w_1:n-1) \approx P(w_n)$

$$P(w_n) = \frac{\text{Count}(w_n)}{\#words}$$

**Bigram (2-gram):**  $P(w_n|w_1:n-1) \approx P(w_n|w_{n-1})$

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

**Trigram (3-gram):**  $P(w_n|w_1:n-1) \approx P(w_n|w_{n-2}, w_{n-1})$

$$P(w_n|w_{n-1}, w_{n-2}) = \frac{\text{Count}(w_{n-2}w_{n-1}w_n)}{\text{Count}(w_{n-2}w_{n-1})}$$

$$\text{General MLE for } n\text{-grams: } P(w_i|w_{n-N+1:n-1}) = \frac{\text{Count}(w_{n-N+1:i})}{\text{Count}(w_{n-N+1:n-1})}$$

- **n-Gram models in practice**

- 3-gram, 4-gram, 5-gram models very common
- The larger the n-grams, the more data required



# n-Gram Models — Bigram Example

Example corpus with 3 sentences

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

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

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

# n-Gram Models — Bigram Example

Example corpus with 3 sentences

<s> I am Sam </s>

<s> Sam I am </s>

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

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

$$P("am" | "I") = \frac{\text{Count}("I am")}{\text{Count}("I")} = \frac{2}{3}$$

$$P("Sam" | "am") = \frac{\text{Count}("am Sam")}{\text{Count}("am")} = \frac{1}{2}$$

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

# n-Gram Models — Bigram Example (25,000 Movie Reviews)

$$P(" < s > \textit{i like the story} < /s > ") = ???$$

**Unigram counts:**

i	like	the	story
87,185	19,862	33,0867	11,094

**Bigram counts:**

	i	like	the	story
i	1	693	20	0
like	326	3	1,997	8
the	15	42	148	5171
story	23	16	16	0

# n-Gram Models — Bigram Example (25,000 Movie Reviews)

$$P(" < s > \ i \ like \ the \ story \ < / s > ") = ???$$

Unigram counts:

i	like	the	story
87,185	19,862	33,0867	11,094

Bigram counts:

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0

Bigram probabilities:

	i	like	the	story
i	0.0	0.007949	0.000229	0.0
like	0.016413	0	0.100544	0.000403
the	0.000045	0.000127	0.0	0.015629
story	0.002073	0.001442	0.001442	0.0

Example calculation:

$$P("like" | "i") = \frac{\text{Count}("i \ like")}{\text{Count}("i")} = \frac{693}{87185} = 0.007949$$

# n-Gram Models — Bigram Example (25,000 Movie Reviews)

Bigram probabilities:

	i	like	the	story
i	0.0	0.007949	0.000229	0.0
like	0.016413	0.0	0.100544	0.000403
the	0.000045	0.000127	0.0	0.015629
story	0.002073	0.001442	0.001442	0.0

Not in the table:

$$P("i"|"<s>") = 0.088198$$

$$P("</s>"|"story") = 0.001262$$



$$\begin{aligned}P("<s> i like the story </s>") &= P("i"|"<s>") \cdot \\ &P("like"|"i") \cdot \\ &P("the"|"like") \cdot \\ &P("story"|"the") \cdot \\ &P("</s>"|"story")\end{aligned}$$

$$\begin{aligned}P("<s> i like the story </s>") &= 0.088198 \cdot \\ &0.007949 \cdot \\ &0.100544 \cdot \\ &0.015629 \cdot \\ &0.001262\end{aligned}$$

$$P("<s> i like the story </s>") = 0.00000000139$$

# n-Gram Models — Practical Consideration

- In general

- Each  $P(w_n|w_1:n-1)$  rather small  $\rightarrow \prod_{n=1}^N P(w_n|w_1:n-1)$  very small
- Risk of arithmetic underflow

→ Always use an equivalent logarithmic format

- Logarithm is a strictly monotonic function

$$\begin{aligned} P_1 \cdot P_2 \cdot P_3 \cdot \dots \cdot P_N &\propto \log (P_1 \cdot P_2 \cdot P_3 \cdot \dots \cdot P_N) \\ &= \log P_1 + \log P_2 + \log P_3 \cdot \dots \cdot \log P_N \end{aligned}$$

# Quick Quiz



# In-Lecture Activity (5 mins)





# Outline

- **Language Models**
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - **Challenges**
- **Smoothing**
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- **Evaluating Language Models**

# Handling OOV Words — Closed vs. Open Vocabulary

- Closed vocabulary

- All strings contain words from a fixed vocabulary

→ **No unknown words**

- Open Vocabulary

- Strings may contain words that are not in the vocabulary (OOV words)

- Examples: proper nouns, mismatching context

→ **Counts might be 0** (even for individual words and not just for long(er) sequences of words)

**Movie review dataset — Unigram counts:**

i	like	the	story	costner	einstein	planck	biden	integral	adverb	tensor	nlp
87,185	19,862	33,0867	11,094	67	20	0	0	27	0	0	0

# Handling OOV Words — Alternatives

- **Special token for OOV words**
  - During normalization, replace all OOV words with a special token (e.g., <UNK>)
  - Estimate counts and probabilities for sequences involving <UNK> like for regular word
- **Subword tokenization** (e.g., with Byte-Pair Encoding)
  - Split texts into tokens smaller than words
  - Tokens are more likely to be frequent
- **Smoothing**

# Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - **Laplace Smoothing**
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- Evaluating Language Models

# Smoothing

- Basic idea

- Avoid assigning probabilities of 0 to unseen n-grams
- "Move" some probability mass from more frequent n-grams to unseen n-grams
- Also called: **discounting**

- Basic method: **Laplace Smoothing** (also: Add-1 Smoothing)

- Example for bigrams

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0



	i	like	the	story
i	1	694	21	1
like	327	1	1,998	9
the	16	43	1	5,172
story	24	17	17	1

# Smoothing — Laplace Smoothing

- Calculating the probabilities

$$\begin{aligned} P_{Laplace}(w_n|w_{1:n-1}) &= \frac{Count_{Laplace}(w_{1:n-1}w_n)}{\sum_w Count_{Laplace}(w_{1:n-1}w)} \\ &= \frac{Count(w_{1:n-1}w_n) + 1}{\sum_w [Count(w_{1:n-1}w) + 1]} \\ &= \frac{Count(w_{1:n-1}w_n) + 1}{Count(w_{1:n-1}) + V} \end{aligned}$$

e.g., for bigrams: 
$$P_{Laplace}(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n) + 1}{Count(w_{n-1}) + V}$$

# Smoothing — Laplace Smoothing

- Effects of smoothing on probabilities

Bigram probabilities (without Laplace Smoothing):

	i	like	the	story
i	0.0	0.007949	0.000229	0.0
like	0.016413	0	0.100544	0.000403
the	0.000045	0.000127	0.0	0.015629
story	0.002073	0.001442	0.001442	0.0

Bigram probabilities (with Laplace Smoothing):

	i	like	the	story
i	0.000006	0.004075	0.000123	0.000006
like	0.003175	0.000010	0.019401	0.000087
the	0.000039	0.000104	0.000002	0.012493
story	0.000255	0.000180	0.000180	0.000011

- Observations

- No zero probabilities (duh!)
- Some non-zero probabilities have changed quite a bit!
- For some n-grams: (arguably) too much probability gets moved to zero probabilities

# Smoothing — Laplace Smoothing

- Effects of smoothing on counts

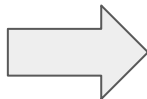
- Question: What counts — without smoothing — would yield  $P_{Laplace}(w_i|w_{i-1})$  ?

$$P_{Laplace}(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}w_n) + 1}{\text{Count}(w_{n-1}) + V} = \frac{\text{Count}^*(w_{n-1}w_n)}{\text{Count}(w_{n-1})}$$

$$\rightarrow \text{Count}^*(w_{n-1}w_n) = (\text{Count}(w_{n-1}w_n) + 1) \cdot \frac{\text{Count}(w_{n-1})}{\text{Count}(w_{n-1}) + V}$$

Bigram counts (original):

	i	like	the	story
i	0	693	20	0
like	326	0	1,997	8
the	15	42	0	5,171
story	23	16	16	0



Bigram counts (adjusted):

	i	like	the	story
i	0.51	355.28	10.75	0.51
like	63.07	0.19	385.34	1.74
the	12.79	34.37	0.80	4133.5
story	2.83	2.00	2.00	0.12



# Smoothing — Laplace Smoothing

- Laplace Discount

- $d_c$  — ratio of adjusted counts to the original counts
- Only defined where original counts > 1

$$d_c = \frac{\text{Count}^*(w_{n-1}w_n)}{\text{Count}(w_{n-1}w_n)}$$

Laplace discounts:

	i	like	the	story
i		0.51	0.54	
like	0.19		0.19	0.22
the	0.85	0.82		0.80
story	0.12	0.13	0.13	

# Add- $k$ Smoothing

- Generalize Laplace (Add-1) Smoothing
  - Add  $k$  instead of 1
  - Set  $0 < k \leq 1$

$$P_{add-k}(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n) + k}{Count(w_{n-1}) + kV}$$

# Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - **Backoff & Interpolation**
  - Kneser-Ney Smoothing
- Evaluating Language Models

# Backoff & Interpolation

- Intuition: Utilize less context if required
  - Assume we want to calculate  $P(w_n|w_{n-2}, w_{n-1})$  but trigram  $w_{n-2}w_{n-1}w_n$  is not in the dataset

## (1) Backoff

- Make use of bigram probability  $P(w_n|w_{n-1})$
- If still insufficient, use unigram probability  $P(w_n)$

## (2) Interpolation

- Estimate  $P(w_n|w_{n-2}, w_{n-1})$  as a weighted mix of trigram, bigram, and unigram probabilities
- Learn weights  $\lambda_i$  from data
- In practice better than Backoff

# Linear Interpolation (example for trigrams)

- Simple interpolation

$$\hat{P}(w_n|w_{n-2}, w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2}, w_{n-1}) \quad \text{with } \sum_i \lambda_i = 1$$

- $\lambda_i$  conditional on context

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

# Backoff & Interpolation

- Learn weights  $\lambda_i$  from data — basic idea
  - (1) Collect held-out corpus
    - Additional corpus *or*
    - Split from initial corpus
  - (2) Calculate all n-gram probabilities
    - Calculation must no consider held-out corpus!
  - (3) Find  $\lambda_i$  that maximize  $\hat{P}(w_n|w_{n-2}, w_{n-1})$  over held-out corpus
    - e.g., using Expectation-Maximization (EM) algorithm (not further discusses here)

# Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- **Smoothing**
  - Laplace Smoothing
  - Backoff & Interpolation
  - **Kneser-Ney Smoothing**
- Evaluating Language Models

# Kneser-Ney Smoothing

- Idea of Kneser-Ney Smoothing: **Absolute Discounting Interpolation**

Remove a fixed value  
from all bigram counts

Interpolation but with better  
estimates for unigram probabilities

$$P_{KN}(w_n|w_{n-1}) = \frac{\max [Count(w_{n-1}w_n) - d, 0]}{Count(w_{n-1})} + \lambda(w_{n-1})P_{KN}(w_n)$$

**Note:** We only look at a bigram language model in the following to keep the examples and notations easy. Kneser-Ney Smoothing analogously defined for larger n-grams.



# Kneser-Ney Smoothing — Absolute Discounting


- Absolute discounting

- Remove fixed value  $d$  from bigram counts  
(typically:  $0 < d < 1$ )
- Makes probability mass for unigrams available
- Intuition

If  $Count(w_{n-1}w_n)$  is large, count hardly affected

If  $Count(w_{n-1}w_n)$  is small, count not that useful to begin with

just a fail-safe to avoid  
negative probabilities


$$\frac{\max[Count(w_{n-1}w_n) - d, 0]}{Count(w_{n-1})}$$

→ Question: How to pick the value(s) for  $d$  ?

# Kneser-Ney Smoothing — Absolute Discounting

- Approach by Church and Gale (1991)
  - Compute bigram counts over large training corpus
  - Compute the counts of the same bigrams over a large test corpus
  - Compute the average count from the test corpus w.r.t. the count in the training corpus

On average, a bigram that occurred 5 times in the training corpus occurred 4.21 times in the test corpus

Bigram count in training corpus	Bigram count in test corpus
0	0.000270
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

→ Set  $d = 0.75$  (maybe a bit smaller for counts of 1 and 2)

# Kneser-Ney Smoothing — Interpolation with a Twist

- Motivation

$$P_{KN}(w_n|w_{n-1}) = \frac{\max [Count(w_{n-1}w_n) - d, 0]}{Count(w_{n-1})} + \lambda(w_{n-1})P(w_n)$$

Using basic interpolation, that would just be the unigram probability

→ But is this actually a good idea?

Predict the missing word:

*"I can't see without my reading \_\_\_\_\_"*

{  
...  
"glasses"  
...  
...  
"Kong"  
...  
}

If *"Hong Kong"* is very frequent:

$$P("Kong") > P("glasses")$$

# Kneser-Ney Smoothing — Interpolation with a Twist

- The difference between *"glasses"* and *"Kong"* — Intuition

- *"glasses"* is preceded by many other words
- *"Kong"* almost only preceded by *"Hong"*

→  $P(w) = \text{"How likely is } w \text{?"}$  maybe not most intuitive approach

- **Alternative:**  $P_{KN}(w) = \text{"How likely is } w \text{ to appear as a novel continuation?"}$

- $P_{KN}(w)$  is high  $\Leftrightarrow$  there are many words  $w'$  that form an existing bigram  $w'w$
- $P_{KN}(w)$  is low  $\Leftrightarrow$  there are only few words  $w'$  that form an existing bigram  $w'w$


→ How can we quantify this?

# Kneser-Ney Smoothing — Interpolation with a Twist


- Calculating  $P_{KN}(w)$

$$P_{KN}(w) = \frac{|\{w' : \text{Count}(w'w) > 0\}|}{|\{(u, v) : \text{Count}(uv) > 0\}|}$$

#words  $w'$  that form an  
existing bigram  $w'w$



total number of existing bigrams  
normalization to ensure that  $\sum_{n=1}^N P(w_n) = 1$



# Kneser-Ney Smoothing — Wrapping it Up

$$P_{KN}(w_n|w_{n-1}) = \frac{\max [Count(w_{n-1}w_n) - d, 0]}{Count(w_{n-1})} + \underbrace{\lambda(w_{n-1})}_{\text{last missing puzzle piece}} P_{KN}(w_n)$$

- Normalizing factor  $\lambda$

- Required to account for the probability mass we have discounted

$$\lambda(w_{n-1}) = \underbrace{\frac{d}{Count(w_{n-1})}}_{\text{normalized discount}} \cdot \underbrace{|\{w' : Count(w_{n-1}w') > 0\}|}_{\substack{\text{\#words that can follow} \\ = \text{\#words that have been discounted} \\ = \text{\#times the normalized discount has been applied}}}$$

# In-Lecture Activity (5 mins)



# Outline

- Language Models
  - Motivation
  - Sentence Probabilities
  - Markov Assumption
  - Challenges
- Smoothing
  - Laplace Smoothing
  - Backoff & Interpolation
  - Kneser-Ney Smoothing
- Evaluating Language Models



# Evaluating Language Models

- A Language Model (LM) is considered good if
  - It assigns high probabilities to frequently occurring sentences
  - It assigns low probabilities to rarely occurring sentences
- 2 basic approaches to compare LMs

## Extrinsic Evaluation

- Requires a downstream task  
(e.g., spell checker, speech recognition)
- Run downstream task with each LM and compare the results
- Can be very expensive & time-consuming

## Intrinsic Evaluation

- Evaluate each LM on a test corpus
- Generally cheaper & faster
- Require intrinsic metric to compare LMs
  - **Perplexity** (among other metrics)

# Intrinsic Evaluation

- 3 core steps for an intrinsic evaluation

- (1) Train LM on a **training corpus**

(i.e., compute the n-gram probabilities)

- (2) Tune parameters of LM using a **development corpus**

(e.g.,  $k$  in case of Add- $k$  Smoothing)

- (3) Compute evaluation metric on **test corpus**

(e.g., perplexity)

- Common corpus breakdown: 80/10/10 (80% training, 10% development, 10% test)

# Perplexity

- Perplexity — Definition

- Inverse probability of test corpus  $W$
- Normalized by the number of words  $N$  in test corpus

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

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

chain rule:

$$= \sqrt[N]{\prod_{n=1}^N \frac{1}{P(w_n | w_1, \dots, w_{n-1})}}$$

e.g., for bigrams:

$$= \sqrt[N]{\prod_{n=1}^N \frac{1}{P(w_n | w_{n-1})}}$$

**Minimizing perplexity  $\Leftrightarrow$  Maximizing probability**

# Perplexity — Intuition

- When is the perplexity **high**?

Many n-grams are frequent in the training corpus but rare in the test corpus



Very few high  $P(w_n|w_{n-1})$  values over test corpus



High perplexity  $PP(W) = \sqrt[N]{\prod_{n=1}^N \frac{1}{P(w_n|w_{n-1})}}$

Many n-grams are rare in the training corpus but frequent in the test corpus



Many low  $P(w_n|w_{n-1})$  values over test corpus



# Perplexity — Practical Consideration

- In general

- Each  $P(w_n|w_1:n-1)$  rather small  $\rightarrow \prod_{n=1}^N P(w_n|w_1:n-1)$  very small
- Risk of arithmetic underflow

- Again, logarithm to the rescue

$$PP(W) = e^{\ln PP(W)}$$

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

$$= -\frac{1}{N} \ln \prod_{n=1}^N \frac{1}{P(w_n|w_1, \dots, w_{n-1})}$$

$$= -\frac{1}{N} \sum_{n=1}^N \ln P(w_n|w_1, \dots, w_{n-1})$$

e.g., for bigrams: 
$$= -\frac{1}{N} \sum_{n=1}^N \ln P(w_n|w_{n-1})$$

# Perplexity — Toy Example

- Evaluation setup

- Bigram LM trained over 25k movie reviews
- Small test corpus  $W$  with  $N = 12$

$W = [$   
    " $\langle s \rangle$  i like good movies  $\langle /s \rangle$ ",  
    " $\langle s \rangle$  the story is funny  $\langle /s \rangle$ "  
 $]$

$$PP(W) = \sqrt[N]{\prod_{n=1}^N \frac{1}{P(w_n|w_{n-1})}} = 40.1$$

bigram	P(bigram)
"<s> i"	0.0882
"i like"	0.0079
"like good"	0.0013
"good movies"	0.0062
"movies </s>"	0.0034
"<s> the"	0.0990
"the story"	0.0156
"story is"	0.1138
"is funny"	0.0022
"funny </s>"	0.0081

# Perplexity — Real-World Example

- Evaluation setup

- Unigram, Bigram, Trigram LMs trained over *Wall Street Journal* articles
- Training corpus: ~38 million words (~20k unique words)
- Test corpus: ~1.5 million words

	Unigram	Bigram	Trigram
Perplexity	962	170	109

# Quick Quiz





# Summary

- Language Models — assigning probabilities to sentences

- Very important concept for many NLP tasks
- Different methods to compute sentence probabilities  
(here: n-grams; later we come back to them using neural networks)

- n-gram Language Models

- Intuitive training → Maximum Likelihood Estimations
- Main consideration: zero probabilities due to large n-grams and/or open vocabularies

↑  
**Markov Assumption** to limited  
size of considered n-grams

↖  
Focus here: **Smoothing**  
(maybe with backoff & interpolation)

} In practice, typically a combination  
of these and similar approaches

# Pre-Lecture Activity for Next Week



# Solutions to Quick Quizzes



# Solutions to Quick Quizzes

