

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 ?

European la de	S_{1} : "on guys all I of noticed sidewalk three a sudden standing the"
Example 1:	S ₂ : "all of a sudden I noticed three guys standing on the sidewalk"
Example 2:	S ₁ : "the role was played by an acress across famous for her comedic timing"
	S ₂ : "the role was played by an acress 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 ~ se lence

(1) Probability of a sequence of words W

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

Example: P("remember to submit your assignment")

WL

(2) Probability of an upcoming word w_n

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

Example: $P("assignment" \mid "remember to submit your")$

In this lecture: How to calculate these probabilities?

Language Models — Applications

- Language Models fundamental for many NLP task
 - **Speech Recognition** P("we built this city on rock and roll") > P("we built this city on sausage rolls")
 - **Spelling correction** $P(" \dots has no mistakes") > P(" \dots has no <u>mistakes"</u>)$
 - **Grammar correction** $P(" \dots has improved") > P(" \dots has improve")$
 - Machine Translation P("I went home") > P("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)

P("*remember to submit your assignment*")

P("assignment" | "remember to submit your")

→ How to calculate those probabilities?

- Quick review: Chain Rule (allows the iterative calculation of joint probabilities)
 - Chain rule for 2 random events:
 - Chain rule for 3 random events:

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

$$P(A_1, A_2, A_3) = P(A_3|A_1, A_2) \cdot P(A_1, A_2)$$

$$= P(A_3|A_1, A_2) \cdot P(A_2|A_1) \cdot P(A_1)$$

■ ...

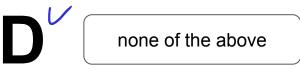
Probabilities of Sentences

• Chain rule — generalization to *N* random events

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

1:5

Quick Quiz $\begin{pmatrix} 6-sided \\ p(l)=1 \\ g(l)=1 \\ p(l)=1 \\ g(l)=1 \\ g(l)=$ $P(A_1) > P(A_1 | A_2)$ P(2) eve)= 1/2 P(2|odu) = O $P(A_1) < P(A_1 | A_2)$ Given two random events A_1 and A_2 with known probabilities $P(A_1)$ and $P(A_2)$, which $P(A_1) = P(A_1 | A_2)$ statement on the right is **always** correct?



Probabilities of Sentences

(and of total probability

$$P(A) = \sum_{n=1}^{\infty} P(An B_{n})$$

• Calculating the probabilities using Maximum Likelihood Estimations

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

$$(and (this is tree)$$

$$(and (this is tree)$$

$$(and (this is tree)$$

$$(i.e., it must be followed by a word)$$

$$(ond (tries is great) \in (i.e., it must be followed by a word)$$

$$(ond (tries is remained) + (i.e., it must be followed by a word)$$

$$(ond (tries is remained) + (i.e., it must be followed by a word)$$

$$(ond (tries is remained) + (i.e., it must be followed by a word)$$

$$(ond (tries is remained) + (i.e., it must be followed by a word)$$

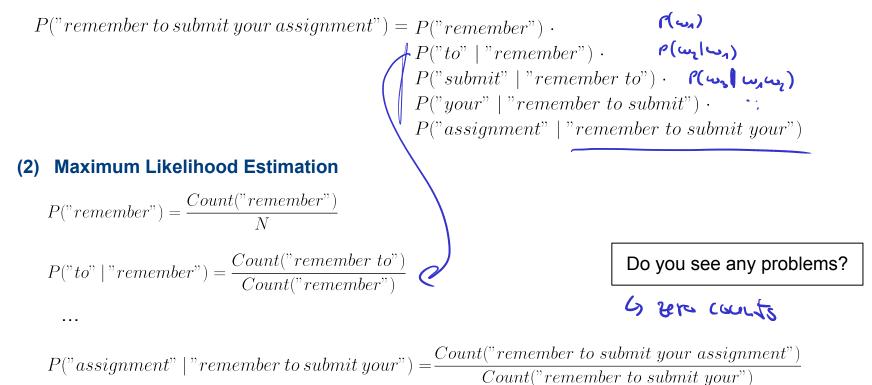
$$(ond (tries is remained) + (i.e., it must be followed by a word)$$

$$(ond (tries is remained) + (i.e., it must be followed by a word)$$

Quick quiz: \ denominator si

Probabilities of Sentences — Example

(1) Application of Chain Rule



Probabilities of Sentences — **Problems**

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

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

$$\rightarrow Count(w_{i:j}) = 0 \quad \Rightarrow \quad \prod_{n=1}^{N} P(w_n | w_{1:n-1}) = 0$$

(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

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

• For our example:

 $P("assignment" \mid "remember to submit your") \approx P("assignment" \mid "your") \qquad \mathbf{2} = \mathbf{1}$ $\approx P("assignment" \mid "submit your") \qquad \mathbf{4} = \mathbf{1}$ $\approx P("assignment" \mid "to submit your") \qquad \mathbf{4} = \mathbf{1}$

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):

$$P(w_n|w_{1:n-1}) \approx P(w_n)$$
 $P(w_n) = \frac{Count(w_n)}{\#words}$

 Bigram (2-gram):
 $P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$
 $P(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n)}{Count(w_{n-1})}$

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

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

- 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

P(

Example corpus with 3 sentences

 Image: Solution of the second state of the second state

$$P("I"|" < s > ") = \frac{Count(" < s > I")}{Count(" < s > ")} = \frac{7}{3}$$

$$P("am"|"I") = \frac{Count("I am")}{Count("I")} = \frac{7}{3}$$

$$P("Sam"|"am") = \frac{Count("am Sam")}{Count("am")} = \frac{7}{3}$$

$$" < /s > "|"Sam") = \frac{Count("Sam < /s > ")}{Count("Sam")} = \frac{7}{3}$$

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>

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

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

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

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

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	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:

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

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 > ") = 0.088198P(" < /s > " | "story") = 0.001262

Quick quiz: Why don't we need P(" < s > ")?

$$P(" < s > i \ like \ the \ story \ ") = \begin{pmatrix} (' < s > ") \\ P("i" |" < s > ") \\ P("ike" |"i") \\ P("ike" |"i") \\ P("the" |"like") \\ P("story" |"the") \\ P(" < s > "|"story") \end{pmatrix}$$

$$P(" < s > i \ like \ the \ story \ ") = 0.088198 \\ 0.100544 \\ 0.0015629 \\ 0.001262 \end{bmatrix}$$

 $P(" < s > i \ like \ the \ story \ </s > ") = 0.0000000139$

n-Gram Models — Practical Consideration

- In general
 - Each $P(w_n|w_{1:n-1})$ rather small $\rightarrow \prod P(w_n|w_{1:n-1})$ very small

N

n=1

Risk of arithmetic underflow

→ Always use an equivalent logarithmic format

Logarithm is a strictly monotonic function

$$P_1 \cdot P_2 \cdot P_3 \cdot \ldots P_N \propto \log (P_1 \cdot P_2 \cdot P_3 \cdot \ldots P_N)$$
$$= \log P_1 + \log P_2 + \log P_3 \cdot \ldots \log P_N$$

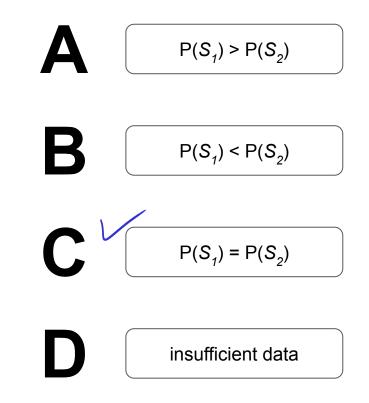
(og(ab) = loga + logb

Quick Quiz

Given a **unigram** language model and the following two sentences S_1 and S_2

- S₁: "alice saw the accident"
- S_{2} : "the accident alice saw"

which sentence has the higher probability?



24

In-Lecture Activity (5 mins) - Show

R'alice saw')

- Task: Calculate the Probability P(saw|alice) given the table of bigram counts below
 - Post your solution to Canvas > Discussions

(individually or as a group; include all group members' names in the post)

P(sau | A(ia) = (and (alice an))=2 alice accident 5 (un) (chice sou) + 70) Coul (cline 100) + 15) Coul (alice acc.) 5) 5 saw alice 15 alice the 20 alice saw saw the 25 accident saw 1 accident alice 2 alico </s> 2

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 OVV 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

Add 1

- Also called: discounting
- Basic method: Laplace Smoothing (also: Add-1 Smoothing)
 - Example for bigrams Count (i stary)= 0

	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	69 <mark>4</mark>	21	1
like	32 <mark>7</mark>	1	1,99 <mark>8</mark>	9
the	1 <mark>6</mark>	4 <mark>3</mark>	1	5,17 <mark>2</mark>
story	2 <mark>4</mark>	1 7	1 7	1

• Calculating the probabilities

$$P_{Laplace}(w_{n}|w_{1:n-1}) = \frac{Count_{Laplace}(w_{1:n-1}w_{n})}{\sum_{w} Count_{Laplace}(w_{1:n-1}w_{n})}$$
$$= \frac{Count(w_{1:n-1}w_{n})(+1)}{\sum_{w} [Count(w_{1:n-1}w_{n})(+1)]}$$
$$= \frac{Count(w_{1:n-1}w_{n})(+1)}{Count(w_{1:n-1})(+1)}$$

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

• Effects of smoothing on 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

Bigram probabilities (without Laplace Smoothing):

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

• Effects of smoothing on counts

Bigram counts (original):

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

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

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

	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			

- Laplace Discount
 - d_c ratio of adjusted counts to the original counts
 - Only defined where original counts > 1

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

	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	

Laplace discounts:

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 if 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 λ_i from data
- In practice better than Backoff

Linear Interpolation (example for trigrams)

• Simple interpolation

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

• λ_i conditional on context

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

Backoff & Interpolation

- Learn weights λ_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 λ_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

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

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

 \rightarrow 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	
~ (), 75	4	3.23	
	5	4.21	
n the st corpus	6	5.23	
	7	6.21	
	8	7.21	
	9	8.26	
	L		

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

Source: <u>A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams</u> (Church and Gale, 1991)

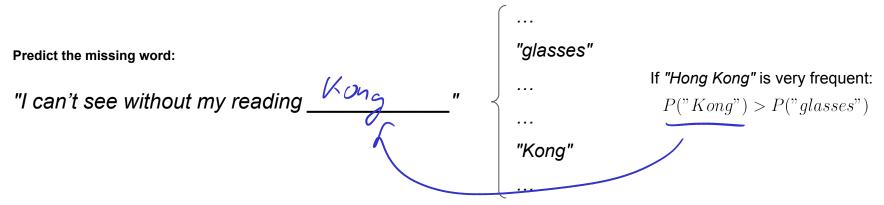
Kneser-Ney Smoothing — Interpolation with a Twist

Motivation

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

Using basic interpolation, that would just be the unigram probability

→ But is this actually a good idea?



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"

Hang Kay King Kang

reading yearse

- \rightarrow P(w) = "How likely is w ?" maybe not most intuitive approach
- Alternative: $P_{KN}(w) =$ "How likely is w to appear as a novel continuation?"
 - $P_{KN}(w)$ is high \Leftrightarrow there are <u>many words</u> w' that form an existing bigram $w'w_{glasses}$
 - $P_{KN}(w)$ is low \Leftrightarrow there are <u>only few words</u> w' that form an existing bigram w'w
 - → How can we quantify this?

Kneser-Ney Smoothing — Interpolation with a Twist

• Calculating $P_{KN}(w)$

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

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

the variation of the st farms and

Kneser-Ney Smoothing — Wrapping it Up

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

- Normalizing factor λ
 - Required to account for the probability mass we have discounted

$$\lambda(w_{n-1}) = \underbrace{\frac{d\ell}{Count(w_{n-1})}}_{\text{normalized discount}} \cdot \underbrace{|\{w': Count(w_{n-1}w') > 0\}|}_{\text{#words that can follow}}$$

= #times the normalized discount has been applied

In-Lecture Activity (5 mins) + 6 redak

- Task: find 5+ words where you would expect that $P_{KN}(w) \ll P(w)$
 - Post your solutions to Canvas > Discussions

 (individually or as a group; include all group members' names in the post)
 - We already used "Kong" as an example, so try to avoid "Francisco", "Angeles", "Aires", etc. :)
 - Optional: Think about how the context matters (e.g., travel blogs vs. movie reviews)

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

 $\Lambda \times$

- 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
 - $\bullet \quad \text{Inverse probability of test corpus } W$
 - Normalized by the number of words N in test corpus

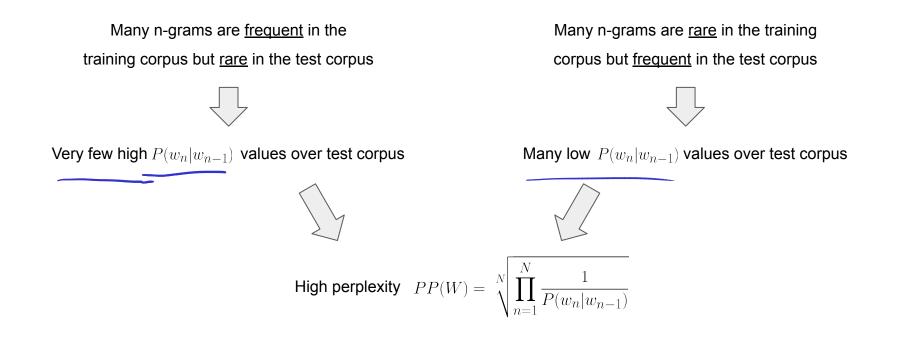
$$PP(W) = P(w_1, w_2, \dots, w_N)^{-\frac{1}{N}}$$

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

Minimizing perplexity ⇔ Maximizing probability

Perplexity — Intuition

- 600
- When is the perplexity high?



Perplexity — **Practical Consideration**

- In general
 - Each $P(w_n|w_{1:n-1})$ rather small $\rightarrow \prod P(w_n|w_{1:n-1})$ very small

N

n=1

- Risk of arithmetic underflow
- Again, logarithm to the rescue

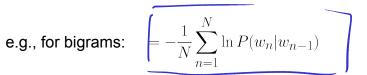
 $PP(W) = e^{\ln PP(W)}$

 $\chi = e^{l_{4}\chi}$

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

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

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



Perplexity — Toy Example

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

 $W = \begin{bmatrix} & & & \\ & "\langle s \rangle & i \ like \ good \ movies \ \langle /s \rangle", \\ & "\langle s \rangle & the \ story \ is \ funny \ \langle /s \rangle" \end{bmatrix}$

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

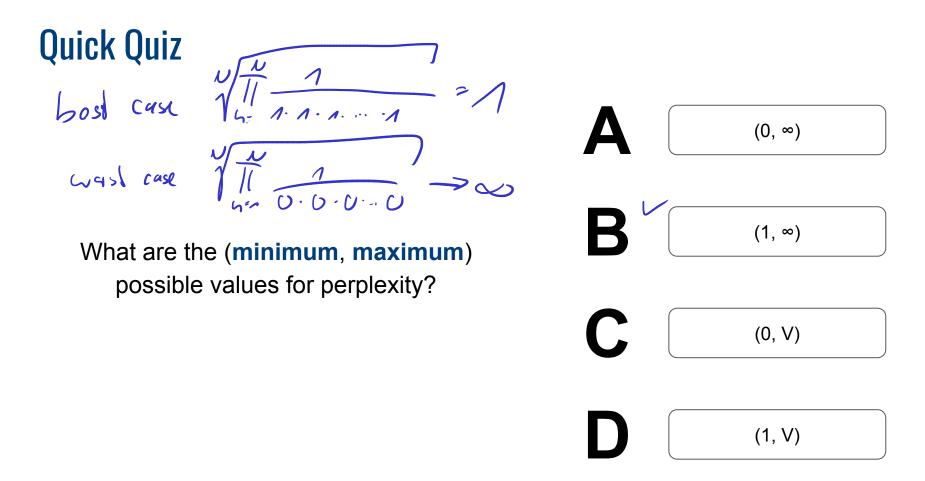
Trigram $LM \rightarrow PP(\omega) = ?$

bigram	P(bigram)		
" <s> i"</s>	0.0882		
"i like"	0.0079		
"like good"	0.0013		
"good movies"	0.0062		
"movies "	0.0034		
" <s> the"</s>	0.0990		
"the story"	0.0156		
"story is"	0.1138		
"is funny"	0.0022		
"funny "	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	4. gram
Perplexity	962	170	109	120
				7
			maybe	



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

- Assigned Task
 - Post a 1-2 sentence answer to the following question into the L2 Discussion on Canvas

"When we want to evaluate classifiers, why is accuracy alone often not a good metric?"

Side notes:

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers but his won't help you learn better

Solutions to Quick Quizzes

- Slide 9: D
 - You can always find examples that violate all other answer options
 - Example: 6-sided die calculate P(2 | "odd") vs P(2 | "even")
- Slide 10
 - We sum all all counts for w_{1:n} followed by any word *w* from the vocabulary
 - If $w_{1:n}w$ does not exist we add 0, so it does not "harm" the total count
- Slide 21:
 - P(" < S > ") is the same for sentences
 - Just for comparing the probabilities of 2 sentences, we don't need it
- Slide 23: D
 - The unigram language model does not care about word order
 - S_1 and S_2 are identical when we ignore the word order

Solutions to Quick Quizzes

- Slide 56
 - Minimum: 1 → All n-gram probabilities are 1
 - Maximum: ∞ → All n-gram probabilities are 0 (or go towards 0)