NUS | Computing
National University of Singapore

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 ~~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

(1) Probability of a sequence of words W

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

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

(2) Probability of an upcoming word w_n

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

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

In this lecture: How to calculate these probabilities?

Language Models — Applications

- Language Models fundamental for many NLP task
  - **Speech Recognition** $P(\text{"}we\ built\ this\ city\ on\ rock\ and\ roll\text{"}) > P(\text{"}we\ built\ this\ city\ on\ sausage\ rolls\text{"})$
  - **Spelling correction** $P(\text{"}\ldots\ has\ no\ mistakes\text{"}) > P(\text{"}\ldots\ has\ no\ \underline{mistaek}\text{"})$
  - **Grammar correction** $P(\text{"}\ldots\ has\ improved\text{"}) > P(\text{"}\ldots\ has\ \underline{improve}\text{"})$
  - **Machine Translation** $P(\text{"}I\ went\ home\text{"}) > P(\text{"}I\ went\ to\ home\text{"})$

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)

$$\left.\begin{aligned} P(\text{"}remember\ to\ submit\ your\ assignment\text{"}) \\ P(\text{"}assignment\text{"} \mid \text{"}remember\ to\ submit\ your\text{"}) \end{aligned}\right\}$$

➜ 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

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

$i:j$ — sequence notations

➜ Chain rule applied to sequences of words

$$P(w_1, \ldots, w_N) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_{1:\,2}) \cdot \ldots \cdot P(w_N|w_{1:\,N-1})$$
$$= \prod_{i=1}^{N} P(w_i|w_{1:\,i-1})$$

Quick Quiz

Given two random events A_1 and A_2 with known probabilities $P(A_1)$ and $P(A_2)$, which statement on the right is **always** correct?

A $P(A_1) > P(A_1 \mid A_2)$

B $P(A_1) < P(A_1 \mid A_2)$

C $P(A_1) = P(A_1 \mid A_2)$

D none of the above

Probabilities of Sentences

- Calculating the probabilities using Maximum Likelihood Estimations

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

Assuming $w_{1:n-1}$ is part for the n-gram
(i.e., it must be followed by a word)

Quick quiz: Why does the denominator simplify like this?

Probabilities of Sentences — Example

(1) Application of Chain Rule

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

(2) Maximum Likelihood Estimation

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

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

$$\ldots$$

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

Do you see any problems?

Probabilities of Sentences — Problems

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

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

(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:

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

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

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

...

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

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>

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

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

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

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

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>

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

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

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

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

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

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

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:

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

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:

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

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

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

$$P("<s>\ i\ like\ the\ story\ </s>") = 0.088198 \cdot 0.007949 \cdot 0.100544 \cdot 0.015629 \cdot 0.001262$$

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

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

n-Gram Models — Practical Consideration

- In general
  - Each $P(w_n|w_{1:\,n-1})$ rather small ➜ $\prod_{n=1}^{N} P(w_n|w_{1:\,n-1})$ very small
  - Risk of arithmetic underflow

➜ Always use an equivalent logarithmic format
- Logarithm is a strictly monotonic function

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

Quick Quiz

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

S_1: *"alice saw the accident"*

S_2: *"the accident alice saw"*

which sentence has the **higher probability**?

A $P(S_1) > P(S_2)$

B $P(S_1) < P(S_2)$

C $P(S_1) = P(S_2)$

D insufficient data

In-Lecture Activity (5 mins)

- 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)

alice accident	5
saw alice	5
alice the	15
alice saw	20
saw the	25
accident saw	1
accident alice	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
  - 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

Add 1 →

	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

$$P_{Laplace}(w_n|w_{1:n-1}) = \frac{Count_{Laplace}(w_{1:n-1}w_n)}{\sum_w Count_{Laplace}(w_{1:n-1}w)}$$

$$= \frac{Count(w_{1:n-1}w_n) + 1}{\sum_w [Count(w_{1:n-1}w) + 1]}$$

$$= \frac{Count(w_{1:n-1}w_n) + 1}{Count(w_{1:n-1}) + V}$$

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 (<u>without</u> 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 (<u>with</u> 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 $P_{Laplace}(w_i|w_{i-1})$?

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

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

Bigram counts (<u>original</u>):

	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 (<u>adjusted</u>):

	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

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

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 $0 < k \leq 1$

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

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

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

$$\text{with } \sum_i \lambda_i = 1$$

- λ_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 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**

Remove a fixed value from all bigram counts

Interpolation but with better estimates for unigram probabilities

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

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

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

➜ 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

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

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

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

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

Kneser-Ney Smoothing — Interpolation with a Twist

- Motivation

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

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 ______________"

$\left\{ \begin{array}{l} \ldots \\ \textit{"glasses"} \\ \ldots \\ \ldots \\ \textit{"Kong"} \\ \ldots \end{array} \right.$

If *"Hong Kong"* is very frequent:

$P(\text{"}Kong\text{"}) > P(\text{"}glasses\text{"})$

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"*

➜ $P(w) =$ *"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 ⇔ there are <u>many words</u> w' that form an existing bigram $w'w$
  - $P_{KN}(w)$ is low ⇔ 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 $P_{KN}(w)$

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

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

total number of existing bigrams

normalization to ensure that $\sum_{n=1}^{N} P(w_n) = 1$

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

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

normalized discount

#words that can follow

= #words that have been discounted

= #times the normalized discount has been applied

In-Lecture Activity (5 mins)

- Task: find 5+ words where you would expect that $P_{KN}(w) < P(w)$
  - 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

- 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

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

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

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

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

Minimizing perplexity ⇔ 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

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

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

Perplexity — Practical Consideration

- In general
  - Each $P(w_n|w_{1:\,n-1})$ rather small ➜ $\prod_{n=1}^{N} P(w_n|w_{1:\,n-1})$ very small
  - Risk of arithmetic underflow

- Again, logarithm to the rescue

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

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

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

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

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 $N = 12$

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

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

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

Source: https://web.stanford.edu/~jurafsky/slp3/3.pdf

Quick Quiz

What are the (**minimum**, **maximum**) possible values for perplexity?

A (0, ∞)

B (1, ∞)

C (0, V)

D (1, V)

v = size of vocabulary

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)