NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Motivation)

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: $P\left(\text{“}remember\ to\ submit\ your\ assignment\text{”}\right)$

(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 section: How to calculate these probabilities?

Language Models — Applications

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

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Sentence Probabilities)

Probabilities of Sentences (more generally: sequence of words)

$$P(\text{“}remember\ to\ submit\ your\ assignment\text{”})$$
$$P(\text{”}assignment\text{”}\ |\ \text{”}remember\ to\ submit\ your\text{”})$$

➜ How to calculate those probabilities?

- Quick review: Chain Rule (allows the iterative calculation of joint probabilities)
  - Chain rule for 2 random events: $P(A_1, A_2) = P(A_2|A_1) \cdot P(A_1)$ $= P(A_1|A_2) \cdot P(A_2)$
  - Chain rule for 3 random events:
$$P(A_1, A_2, A_3) = P(A_3|A_1, A_2) \cdot P(A_1, A_2)$$
$$= P(A_3|A_1, A_2) \cdot P(A_2|A_1) \cdot P(A_1)$$
  - …

Probabilities of Sentences

Unordered

order arbitrary.

- 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

Ordered.

➜ Chain rule applied to sequences of words

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

Probabilities of Sentences

- Calculating the probabilities using Maximum Likelihood Estimations

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

prefix

prefix

word

current word.

Any possible word can follow.

not w_n

Probabilities of Sentences — Example

(1) Application of Chain Rule

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

(2) Maximum Likelihood Estimation

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

total # of words in corpus

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

...

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

Probabilities of Sentences — Problems

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

- 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

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

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

➜ Can we keep the sequences short?

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Markov Assumption)

Markov Assumption

- Probabilities depend on only on the last k words

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

- For our example:

$$P(\text{"assignment"} \mid \text{"remember to submit your"}) \approx P(\text{"assignment"} \mid \text{"your"}) \quad k=1$$

$$P(\text{"assignment"} \mid \text{"submit your"}) \quad k=2$$

$$P(\text{"assignment"} \mid \text{"to submit your"}) \quad k=3$$

...

n-Gram Models

how many token in gram

context

Unigram (1-gram): $P(w_n|w_{1:n-1}) \approx P(w_n)$

k = 0

context independent

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

k = 1

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

k = 2

context dependent

n-Gram Models

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): $P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$

Maximum Likelihood Estimation

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

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

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

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>

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

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

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

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

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>

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

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

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

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

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

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	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:

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

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

Other entries not in the table:

$$P(\text{"}i\text{"}\,|\,\text{"}<s>\text{"}) = 0.088198$$

$$P(\text{"}</s>\text{"}\,|\,\text{"}story\text{"}) = 0.001262$$

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

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

$$P(\text{"}<s>\ i\ like\ the\ story\ </s>\text{"}) = 0.00000000139$$

n-Gram Models — Practical Consideration

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

➜ Always use an equivalent logarithmic format

- Logarithm is a strictly monotonic function

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

Check the options that are correct.

- [ ] k is for the size of the gram (number of tokens)
- [x] n is for the size of the gram (number of tokens)
- [x] k is for the size of the context (number of tokens)
- [ ] n is for the size of the context (number of tokens)

✔

What happens we don't use a special token to mark the end or beginning of a sentence / text?

- [x] We will have slightly fewer grams in our vocabulary.
- [x] We won't be able to distinguish the start or end of a text or input.
- [x] Probability estimates will generally be poorer for short inputs.

✔

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Challenges)

Dealing with Sparsity

- Motivation
  - Let's assume we have a bigram language model based 25,000 movie reviews
  - Task: Compute the probability of the sentence *"i grasp the story"*
    (compare with the previous example sentence *"i love the story"*)

$$P(\text{"}<s>\ i\ grasp\ the\ story\ </s>\text{"}) = P(\text{"}i\text{"}|\text{"}<s>\text{"}) \cdot P(\text{"}grasp\text{"}|\text{"}i\text{"}) \cdot P(\text{"}the\text{"}|\text{"}grasp\text{"}) \cdot P(\text{"}story\text{"}|\text{"}the\text{"}) \cdot P(\text{"}</s>\text{"}|\text{"}story\text{"})$$

$$P(\text{"}<s>\ i\ grasp\ the\ story\ </s>\text{"}) = 0.088198 \cdot 0.0 \cdot 0.226190 \cdot 0.015629 \cdot 0.001262$$

$$P(\text{"}<s>\ i\ grasp\ the\ story\ </s>\text{"}) = 0.0$$

Problem: our dataset to train the language model did not contain the bigram *"i grasp"*.

$$Count(\text{"}i\ grasp\text{"}) = 0$$

Dealing with Sparsity

- Problem: **zero-counts**
  - n-grams not seen during the training of the language model
  - Very common in practice since our training data will be sparse
    (the likelihood for this naturally increases with the size of the n-grams)

$$P(\text{"}<s>\ i\ grasp\ the\ story\ </s>\text{"}) = 0.0$$

Question: Is it really acceptable to assign a probability of 0 to a perfectly fine sentence?

- Basic idea: avoid zero-counts
  - Assume that any n-gram has a non-zero probability

➜ **Smoothing**

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Smoothing I: Laplace Smoothing)

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

no zero 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

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

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

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

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

vocabulary 全部词汇

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

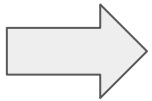
- Effects of smoothing on counts
  - Question: What counts — without smoothing — would yield $P_{Laplace}(w_i|w_{i-1})$?

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

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

Bigram counts (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

Add-k Smoothing

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

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

k is hyper parameter

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Smoothing II: Common Refinements)

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

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

with $\sum_i \lambda_i = 1$

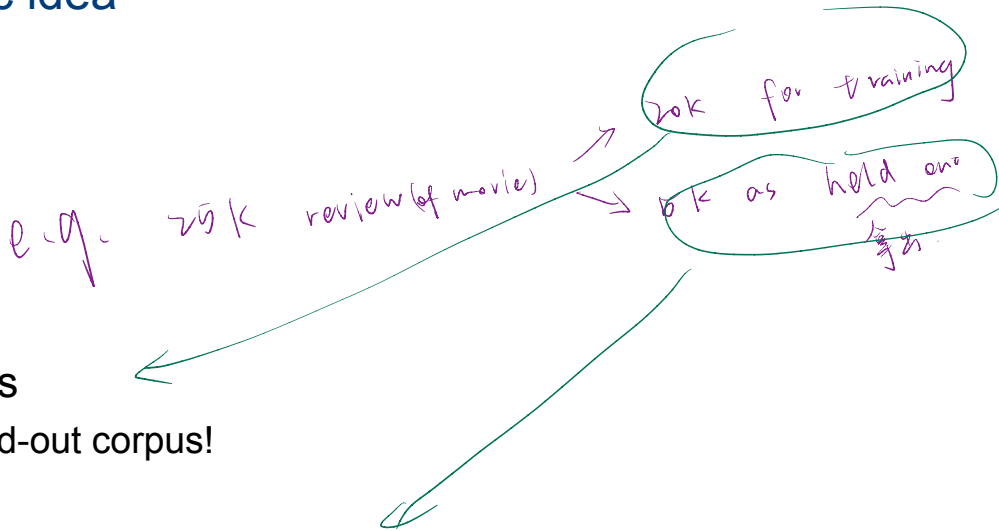
- λ_i conditional on context

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

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)

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Smoothing III: Advanced Techniques)

Kneser-Ney Smoothing

most common model

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

Remove a fixed value from all bigram counts

Interpolation but with better estimates for unigram probabilities

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

0.75

详情看 Stanford PPT.

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 ?

0.75

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 Kong *"*

↑ bad choice.

…
"glasses"
…
…
"Kong"
…

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"* is 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 many words w' that form an existing bigram $w'w$
  - $P_{KN}(w)$ is low ⇔ 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 $P_{KN}(w)$

using counts.

reading glasses.

w' → sun glasses.

#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

new unigram probabilities.

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

$\Sigma P_{KN} = 1$

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

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

= #words that have been discounted

= #times the normalized discount has been applied

NUS | Computing
National University of Singapore

Natural Language Processing: Foundations

Section 3 — n-Gram Language Models (Evaluation)

Evaluating Language Models

用莎士比亚的文章训练
不符合现代人.

- A Language Model (LM) is considered good if
  - It assigns high probabilities to frequently occurring sentences in a corpus
  - It assigns low probabilities otherwise (e.g., rarely occurring sentences)

- 2 basic approaches to compare LMs

外部

Extrinsic Evaluation

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

Intrinsic Evaluation

- Evaluate each LM on a test corpus
- Generally cheaper and faster
- Require an intrinsic metric to compare LMs

  ➜ **Perplexity** (among other metrics)

Intrinsic Evaluation

- 3 core steps for an intrinsic evaluation

  (1) Train the LM on a **training corpus**
  (i.e., compute the n-gram probabilities)

  (2) Tune parameters of the LM using a **development corpus**
  (e.g., *k* in case of Add-*k* Smoothing, *d* in case of Kneser-Ney Smoothing)

  (3) Compute evaluation metrics on a **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

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

geometric mean

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

chain rule:

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

为了简化

e.g., for bigrams:

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

Minimizing perplexity ⇔ Maximizing probability

Perplexity，翻译为中文的意思为：困惑、混乱。在NLP中，表示 语言模型 的困惑度，是交叉熵的指数形式，可以作为语言模型的评价指标。Perplexity越低，说明模型拟合效果越好。Perplexity 计算公式如下：

$$\begin{aligned}\mathrm{Perplexity(S)} &= \mathrm{p(w_1, w_2, w_3, ..., w_m)}^{-1/m} \\ &= \sqrt[m]{\prod_{i=2}^{m} \frac{1}{\mathrm{p(w_i | w_1, w_2, w_3, ..., w_{i-1})}}}\end{aligned}$$

简单来说，perplexity刻画的是语言模型预测一个语言样本的能力，比如已经知道了$(\mathrm{w_1, w_2, w_3, ..., w_m})$这句话会出现在语料库之中，那么通过语言模型计算得到这句话的概率越高，说明语言模型对这个语料库拟合的越好。

在语言模型的训练中，通常采用perplexity的对数表达形式：

$$\log(\mathrm{perplexity(S)}) = -\frac{1}{m}\sum_{i=2}^{m}\mathrm{p(w_1, w_2, w_3, ..., w_m)}$$

在数学上，log perplexity 可以看作真实分布与预测分布之间的 交叉熵 Cross Entropy, 交叉熵描述了两个概率分布之间的一种距离，假设 x 是一个离散变量，$\mathrm{u(x)}$ $\mathrm{v(x)}$ 是两个与 x 相关的概率分布，那么 $\mathrm{u, v}$ 之间的交叉熵的定义是分布 u 下 $-\log(\mathrm{v(x)})$ 的期望值：

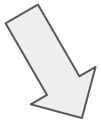
$$\mathrm{H(u, v)} = \mathrm{E_u}\left[-\log(\mathrm{v(x)}\right] = -\sum_{x}\mathrm{u(x)}\log(\mathrm{v(x)})$$

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

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

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

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

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

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 $N = 12$

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

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

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

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

越低，拟合越好。

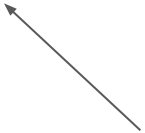
Source: https://web.stanford.edu/~jurafsky/slp3/3.pdf

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

Step 1:

add padding (填料):

把 [["I", "am", "Bob"], [...], ...]

变成 [["<s>", "I", "am", "Bob", "</s>"], ...]

```
padding_text = [ ], sos = '<s>', eos = '</s>'.
for text in tokenized_texts:
    padded_text = [self.sos] + text + [self.eos]
    padding_texts.append(padded_texts).
```

step 2. bwild vocabulary

```
vocabulary = set()
```