N-gram Language Models

Sunkyung Park

Contents

Introduction

- Language Models or LMs
    - Models assign probabilities to sequence of words

- N-gram
    - The simplest model that assign probabilities to sentences and sequences of words.
    - Examples
        - Two-word sequence of words
            - “please turn”, “turn your”, or “your homework”
        - Three-word sequence of words
            - “please turn your”, “turn your homework”
    - Estimate the probability of the last word of an n-gram given the previous words.
    - Assign probabilities to entire sequences.

N-Grams (which looks $n-1$ words into the past)

- Intuition
  - Instead of computing the prob of a word given its entire history
  - Approximate history by just the last few words.
  - The bigram model
    - $\mathrm{P}(w_n|w_{1:n-1})$
    - Sentence: "Walden Pond's water is so transparent that the"
    - (old) $\mathrm{P}(the|Walden\ Pond's\ water\ is\ so\ transparent\ that)$
    - (n-gram) $\mathrm{P}(the|that)$
    - Approximation

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

- Based on Markov assumption
  - The prob of a word depends only on the previous word.
  - The class of probabilistic models
    - we can predict the prob of some future unit without looking too far into the past.

- Probability
  - Notation
    - A word w given a history h or an entire word sequence W
    - $P(w_1, w_2, \dots, w_n)$ = **joint probability** of each word in a sequence.
  - How to compute join probability $P(w_1, w_2, \dots, w_n)$?
    - Decompose this prob using chain rule of probability

Chain Rule of Prob

$$\begin{aligned} P(X_1 \dots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2})\dots P(X_n|X_{1:n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \end{aligned}$$

Applying to words

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\dots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

Based on Markov assumption

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

- Equation

Word

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

- N = 2 means bigrams and N = 3 means trigrams
- Approximate the prob of a word w_n given its entire context($w_{1:n-1}$)

Sequence

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

- Compute the prob of a complete word sequence

N-Grams : how to estimate n-gram prob?

- Intuition : Maximum likelihood estimation(MLE)
  - Example problem
    - Compute bigram probability of a word w_n given a previous word w_{n-1}.

Same

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

(Denominator) Sum of all the bigrams that share the same first word w_{n-1}

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

  - General case

$$\boxed{P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\ w_n)}{C(w_{n-N+1:n-1})}}$$

(numerator) observed frequency of a particular sequence

(denominator) observed frequency of a prefix

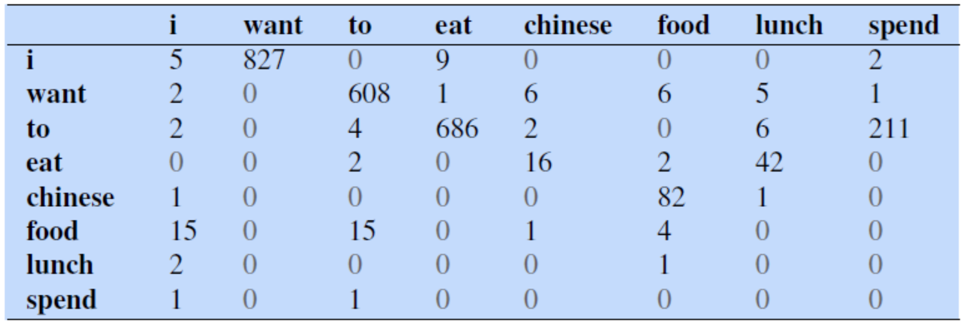
- **Ratio = Relative frequency**
- **Relative frequency is an example of MLE.**

  - In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M($P(T|M)$)
    - Case. The word *Chinese* occurs 400 times in a corpus of a million words.
    - Q. What is the prob that a random word selected from a million words will be the word *Chinese*?
    - A. The Maximum likelihood estimation is .0004
    - The Probability makes it most likely that *Chinese* will occur 400 times in a million-word corpus

N-Grams : Example

- Case. A dialogue system from Now-defunct Berkley Restaurant Project
- Model. Bigram

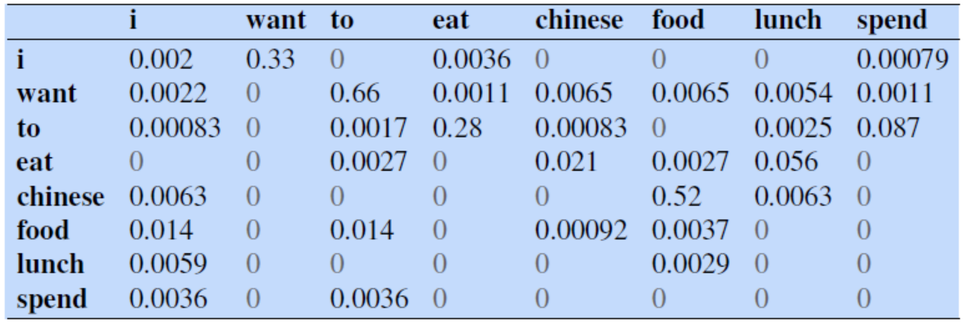
Raw data



	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

Normalize by the unigram for row



	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

- We represent and compute language model probabilities in log format as log probabilities. Why?
  - The more probabilities we multiply together, the smaller the product becomes. ➔ underflow!

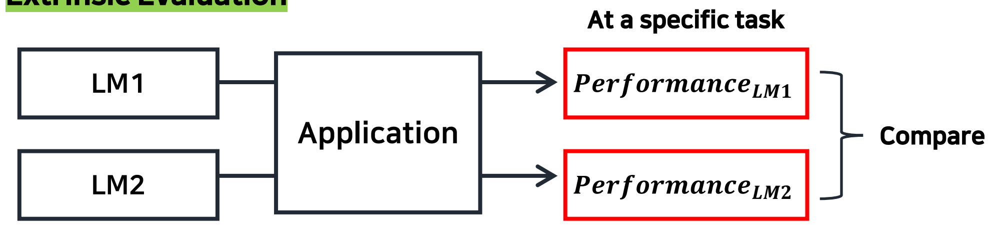
$$\begin{aligned} P(\texttt{<s> i want english food </s>}) &= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\ &\quad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\ &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &= .000031 \end{aligned}$$

  - By log probabilities, we get numbers are not as small.
  - If we need raw probabilities, we can just the exp of the log prob.

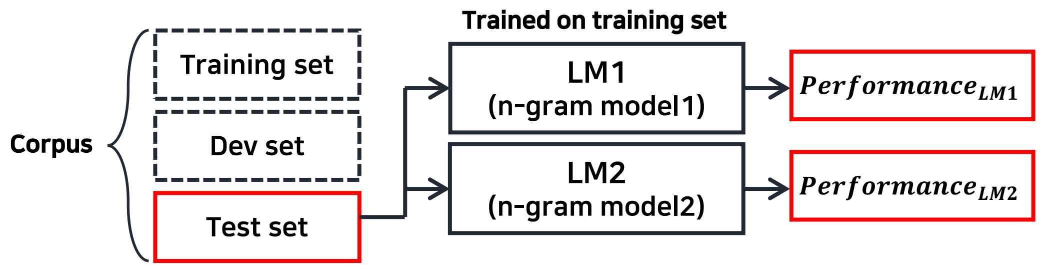
$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

▪ Evaluating Language Models

Extrinsic Evaluation



Intrinsic Evaluation



- Compare how well trained models fit the test sets
  - Better model will better predict test sets in detail.
  - Better model will assign a higher probability to the test

▪ Perplexity(PP for short) (⊂ Intrinsic Evaluation)

- The perplexity of a language model on a test set
  - The inverse probability of the test set, normalized by number of words
  - For a test set $W = w_1 w_2 \dots w_n$

$$\mathrm{PP}(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Minimize perplexity
=
Maximize the test set prob according to LM.

Based on Chain rule

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

Based on Markov Assumption(w/ bigram)

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

- Weighted average branching factor
  - Branching factor: the num of possible next words that can follow any word.
  - At Task(Recognize the digits in English

(Case 1)
IF (in training set and test set)
Prob of each of 10 digits = $\frac{1}{10}$

THEN branching factor = 10 and
Perplexity of test string of digits of length N = 10

(Case 2)
IF (in training set and test set) Zero occurs far more often

IN TEST SET : 0 0 0 0 0 3 0 0 0 0 0

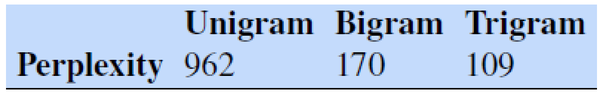
THEN branching factor = 10 but
Perplexity or Weighted branching factor is lower than Case 1.

Evaluating Language Models : Perplexity

- How perplexity can be used to compare different LMs (e.g n-gram models)

Perplexity of models

on WSJ test set of 1.5 million words



	Unigram	Bigram	Trigram
Perplexity	962	170	109

  - **The more information the n-gram gives us about the word sequence, The lower the perplexity.**
  - **(Again) perplexity is related inversely to the likelihood of the test sequence according to the model.**

- Note!
  - In computing perplexity, n-gram model must be constructed without any knowledge of the test.
  - The perplexity of two LMs is only comparable if they use identical vocabularies.
  - An (intrinsic) improvement in perplexity ≠ an (extrinsic) improvement in the performance of a task like speech recognition.

    ➔ Therefore, model's improvement in perplexity should be confirmed by an evaluation of a real task before concluding the model evaluation.

Sampling sentences from a language model

- One way to visualize what kind of knowledge a language model embodies is to sample from it.
- Sampling from a LM means to generate some sentences according to its likelihood defined by the model.
  - It is more likely to generate sentences that the model thinks a high prob.
  - Shannon(1951) and Miller and Selfridge(1950)
    - Unigram case in English language
      - each word covering an interval proportional to its frequency.

Computed from
the text of this book.

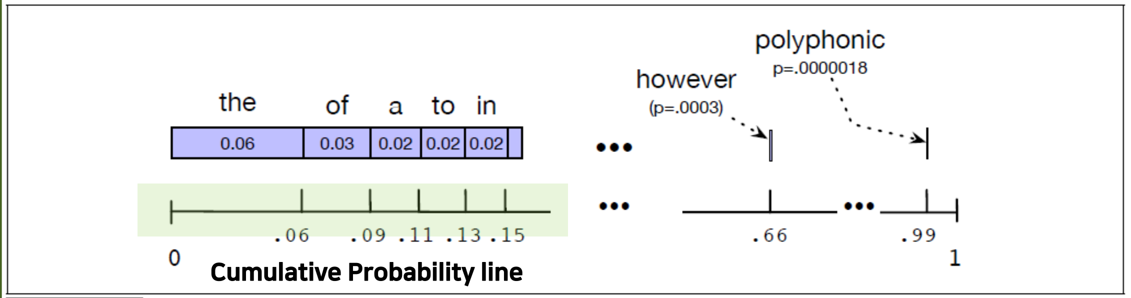


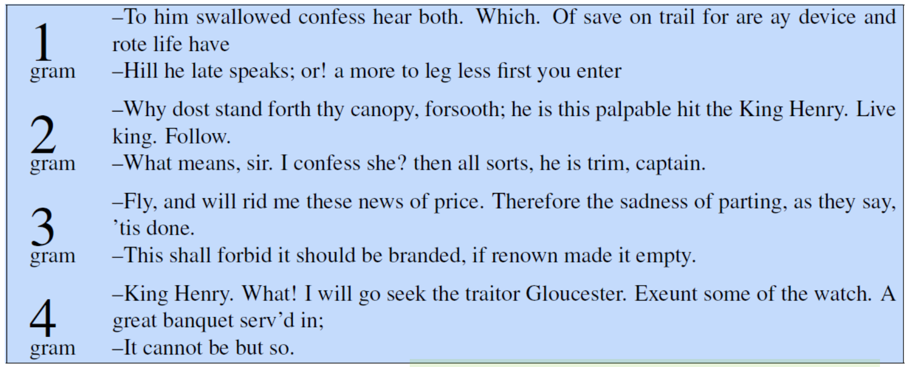
Figure 3.3 A visualization of the sampling distribution for sampling sentences by repeatedly sampling unigrams. The blue bar represents the frequency of each word. The number line shows the cumulative probabilities. If we choose a random number between 0 and 1, it will fall in an interval corresponding to some word. The expectation for the random number to fall in the larger intervals of one of the frequent words (*the*, *of*, *a*) is much higher than in the smaller interval of one of the rare words (*polyphonic*).

[Process]

- Choose random variable between 0 and 1
- Find that point on probability line
- Print(Generate) the word whose interval includes this chosen value
- Continue choosing random numbers and generating words until we generate the sentence-final </s>

Generalization and Zeros

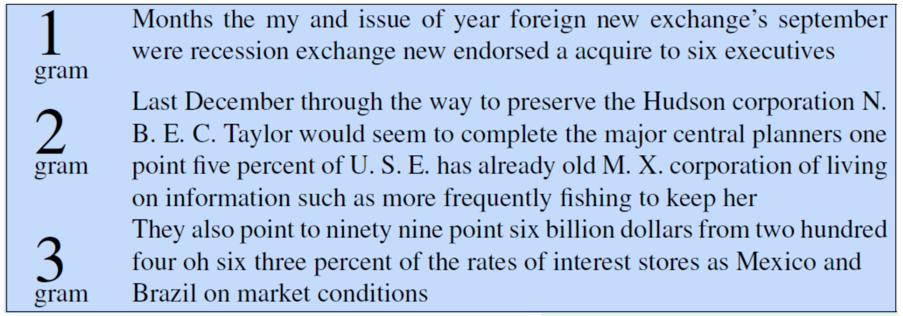
- Let's start with case below…



1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Figure 3.4 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

pseudo-Shakespeare



1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 3.5 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

WSJ

(Finding) No overlap in generated sentences, little overlap in small phrases!

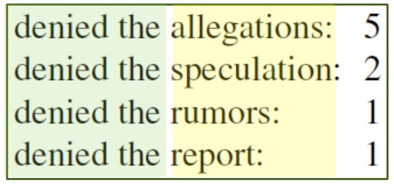
(Problem) Statistical models(n-gram models) are likely to be useless as predictors

if the training set and the test sets are different as Shakespeare and WSJ!

(How to overcome?) Next parts

- How to overcome?
  - Use a training corpus that has similar genre to whatever task we are trying to accomplish.
  - To get training data in the appropriate dialect or variety.
    - $finna$ in African American language ⊂ is going to
    - den in African American language ⊂ then
  - Solutions above are not sufficient. Model has still the problem of sparsity.

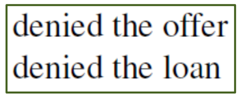
WSJ Treebank3 corpus



denied the	allegations:	5
denied the	speculation:	2
denied the	rumors:	1
denied the	report:	1

● **Words followed by the "denied the"**

Test set



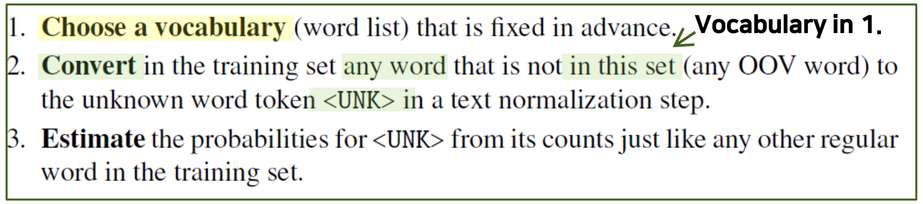
denied the offer
denied the loan

Our model estimate $P(offer|deninde\ th)$ is 0

- Zero are a problem of two reasons?
  - Reason 1:
    - The prob of all words likely to occur would be underestimated.
  - Reason 2:
    - The prob of any word in test set is 0.
    - the entire prob of the test set is 0
    - the perplexity is based on the inverse prob of test set
    - Some words with zero prob, we can't compute perplexity with $\frac{1}{0}$

Generalization and Zeros: Unknown Words

- Unknown words or Out of vocabulary(OOV) words
  - Definition
    - Words never seen before
  - OOV rate
    - Percentage of OOV words that appear in the test set
- Open vocabulary system
  - Model potential unknown words in the test set by adding a pseudo-word called "<UNK>"
- 2 Ways to train the probabilities of model <UNK>
  - To turn the problem back into a closed vocabulary one by choosing a fixed vocabulary in advance.



1. **Choose a vocabulary** (word list) that is fixed in advance. Vocabulary in 1.
2. **Convert** in the training set any word that is not in this set (any OOV word) to the unknown word token <UNK> in a text normalization step.
3. **Estimate** the probabilities for <UNK> from its counts just like any other regular word in the training set.

  - To create such a vocabulary implicitly when we don't have a prior vocabulary in advance.
    - 1st way: replace all words occur fewer than n(small number) times in training set by <UNK>.
    - 2nd way: select a vocabulary size V in advance and choose top V words by frequency and replace the rest by <UNK>.
- <UNK> model affects to perplexity.
  - Low perplexity by choosing a small vocabulary and assigning the unknown word a high probability.
  - Perplexity should be compared across LM w/ same vocabularies.

▪ Smoothing

- Words that are in our voca(not unknow word) but appear in a test set in an unseen context(they never appeared after a word in training, but it is not really.)
- To keep a LM from assigning zero prob to unseen events
  - Shave off a bit of prob mass from more frequent events and Give it to the unseen events.
  - This is called "smoothing" or "discounting".
    - Laplace smoothing, add-k smoothing, stupid backoff, Kneser-Ney smoothing

- Laplace(add-one) smoothing
  - +1 to all the n-gram counts before normalizing them into prob.
  - Not perform well in modern n-grams, but introduces useful concepts in other algorithms, gives a baseline.

● Normalization factor

$$\boxed{P(w_i) = \frac{c_i}{N}} \ggg \boxed{P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}} \ggg \boxed{c_i^* = (c_i+1)\frac{N}{N+V}} \ggg \boxed{d_c = \frac{c^*}{c}}$$

- Unsmoothed MLE of the unigram probability of the word w_i
- c_i: count of w_i
- N: total number of word tokens

- Laplace smoothing
- Numerator: incremented by 1
- Denominator: take into account the extra V

- (Convenient ver.) Laplace smoothing
- Convenient to describe how smoothing algorithm affects the numerator w/ $adjusted\ c^*$.
- Easier to compare with MLE counts and Possible to be turned into a prob by normalizing by N. ($P_i^* = \frac{c_i^*}{N}$)

- Smoothing = a relative discount d_c
- Ratio of the discounted counts to the original counts

▪ Smoothing : Laplace smoothing

Step I. Unsmoothed(Original)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

»

Step II. Add-one smoothed counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 3.6 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

Step III. Smoothed bigram probabilities

$$\boxed{P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}}$$

$$\boxed{P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 3.7 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

»

Step IV. Count matrix how much a smoothing algorithm has changed the original counts.

● Normalization factor

$$\boxed{c_i^* = (c_i + 1)\frac{N}{N + V}}$$

$$\boxed{c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figure 3.8 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero counts are in gray.

- ● $C(want\ to)$: 609 ➔ 238
- ● $P(to|want)$: .66 ➔ .26
- ● d: $\frac{238}{609}$ = .39, ● d: $\frac{8.2}{83}$ = .098

➔ **Sharp change in counts and probabilities occurs because too much probability mass it moved to all zeros.**

Smoothing: Add-k smoothing and Back off, Interpolation

- Add-k smoothing
  - Instead of +1, + a fractional count k(.5, .05, .01)
  - Requirement: a method for choosing k by optimizing on a devset.

  ➔ Does it improve performance? Well…

- Suppose that…
  - We want to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$. What then?

- Back off(way to use n-gram "hierarchy")
  - We only "back off" to a lower-order n-gram with zero evidence for a higher-order n-gram.
    - For example:
      - We can "estimate" its prob by using the bigram prob $P(w_n|w_{n-1})$.
  - Katz back-off(back off with discounting)

Discounted prob

If N-gram have non-zero counts

$$P_{\mathrm{BO}}(w_n|w_{n-N+1:n-1}) = \begin{cases} P^*(w_n|w_{n-N+1:n-1}), & \text{if } C(w_{n-N+1:n}) > 0 \\ \alpha(w_{n-N+1:n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2:n-1}), & \text{otherwise.} \end{cases}$$

f(α) to distribute prob mass to lower order n-grams

Prob of (N-1)-gram

- Interpolation(way to use n-gram "hierarchy")
  - In simple linear interpolation
    - We estimate the trigram prob $P(w_n|w_{n-2}w_{n-1})$ by mixing the unigram, bigram, and trigram probs, each weighted by a λ.

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

A particular trigram

$$\sum_i \lambda_i = 1$$

λ must sum to 1.

➔ The prob estimates is a weighted average

- Interpolation
  - In sophisticated version of linear interpolation(conditional interpolation)
    - Each weight λ is computed by conditioning on the text.
      - We have accurate counts for a particular bigram, then the counts of the trigrams based on this bigram will be more trustworthy.

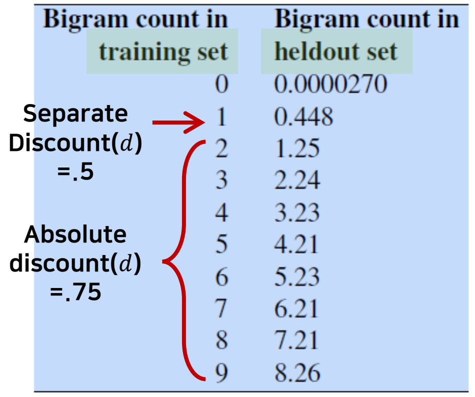
$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2:n-1})P(w_n) \\ &+\lambda_2(w_{n-2:n-1})P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2:n-1})P(w_n|w_{n-2}w_{n-1})\end{aligned}$$

Make λs for those trigrams higher(more weight)

  - How to set λ?
    - Set λ values with held-out corpus(from the training data)
      - Fix the n-gram prob by training data
      - Search for λ values that maximize the likelihood of the held-out corpus

▪ Kneser-Ney Smoothing(most commonly used and best performing)

- (starts with)Absolute discounting
  - Discounting: save prob for smoothing algorithm to distribute to the unseen n-grams.
  - “How much we should discount?”
    - Clever idea(church and gale, 1991) is to look at a held-out corpus



Separate Discount(d) =.5

Absolute discount(d) =.75

Bigram count in training set	Bigram count in heldout set
0	0.0000270
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

- (Training set) A bigram grammar from 22 million words of AP newswire.
- (Held-out set) Checked the counts of each of these bigrams in another 22 million words.
- (Result) a bigram that occurred 4 times in the first 22 million words occurred 3.23 times in next 22 million words.
- (Intuition) since we have good estimates already for the high counts, a small discount won’t affect them much.

- Equation

The discounted bigram

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \boxed{\frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}\,v)}} + \boxed{\lambda(w_{i-1})P(w_i)}$$

The unigram with weight λ

- “Kneser-Ney discounting” augments absolute discounting w/ a way to handle the lower-order unigram distribution.
  - It can give the answer about “How likely w to appear as novel continuation?”

- Equation

$$\boxed{P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}|}$$

$$\boxed{P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(vw') > 0\}|}}$$

● Numerator: the num of word types to precede w

● Denominator: the num of words preceding all words.

- Case: Predict the next word interpolating a bigram and a unigram model.

I can’t see without my reading _______________ .

- *Kong or glasses?*
  - *Kong* : it is frequent but only frequent in the phrase Hong Kong.
  - *Glasses*: it has wider distribution.

➔A frequent(*Kong*) occurring in only one context(*Hong*) is low continuation prob.

Kneser-Ney Smoothing

- **Final equation for interpolated Kneser-Ney smoothing for bigrams**

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\text{CONTINUATION}}(w_i)$$

λ : normalizing constant to distribute the prob mass discounted

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}|$$

● **Normalized discount**

● **The number of word types that can follow w_{i-1}**

● **The number of times we applied the normalized discount**

- **The recursive formulation**

$$P_{\text{KN}}(w_i|w_{i-n+1:i-1}) = \frac{\max(c_{KN}(w_{i-n+1:i}) - d, 0)}{\sum_v c_{KN}(w_{i-n+1:i-1}\ v)} + \lambda(w_{i-n+1:i-1}) P_{KN}(w_i|w_{i-n+2:i-1})$$

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuationcount}(\cdot) & \text{for lower orders} \end{cases}$$

- **Count c_{KN} depends on whether we are counting the highest-order n-gram or one of the lower-order n-grams**
  - **Highest-order: trigram if being interpolating trigram, bigram and unigram.**
  - **Lower-order: bigram or unigram if being interpolating trigram, bigram and unigram.**
    - **Continuation count is the number of unique single word contexts for ·**

Advanced: Perplexity's Relation to Entropy

- Entropy
  - Measure of information (average degree of uncertainty)
- Equation

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x)$$

Given a random variable X(words, letters,⋯, etc.)and a probability function $p(x)$, the entropy of the random variable X is.
Log base 2 means the entropy will be measured in bits.

- Intuition
  - Lower bound on the number of bits to encode a piece of information.
  - Case: On a horse race bet, we want to send a short msg to bookie to tell home which of the 8 horses to bet on.

Distribution of the bets placed

Most probable	horse1	$\frac{1}{2}$	horse5	$\frac{1}{64}$
	horse2	$\frac{1}{4}$	horse6	$\frac{1}{64}$
	horse3	$\frac{1}{8}$	horse7	$\frac{1}{64}$
	horse4	$\frac{1}{16}$	horse8	$\frac{1}{64}$

$H(X)$ = 2bits (sending msg in 2bits)

horse1	$\frac{1}{8}$	horse5	$\frac{1}{8}$
horse2	$\frac{1}{8}$	horse6	$\frac{1}{8}$
horse3	$\frac{1}{8}$	horse7	$\frac{1}{8}$
horse4	$\frac{1}{8}$	horse8	$\frac{1}{8}$

$H(X)$ = 3bits(sending msg in 3 bits)

- The entropy of some sequence of words $W = \{w_1, w_2, \dots, w_n\}$
  - Compute the entropy of a random variable that ranges over all finite sequences of words of length n in some language L.

$$H(w_1, w_2, \dots, w_n) = -\sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

  - Entropy rate(per-word entropy)

$$\frac{1}{n} H(w_{1:n}) = -\frac{1}{n} \sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

- (sequences of infinite length)
  - L: stochastic process that produces a sequence of words
  - W: represent the sequence of words $w_1, w_2, \dots, w_n$
  - $H(L)$: entropy rate

$$\begin{aligned} H(L) &= \lim_{n \to \infty} \frac{1}{n} H(w_1, w_2, \dots, w_n) \\ &= -\lim_{n \to \infty} \frac{1}{n} \sum_{W \in L} p(w_1, \dots, w_n) \log p(w_1, \dots, w_n) \end{aligned}$$

Advanced: Perplexity's Relation to Entropy

- Shannon-McMillan-Breiman theorem
  - Assumption: language is regular in certain ways(stationary and ergodic)
    - Stationary: $Probability\ distribution\ of\ words_{t}$ = $Probability\ distribution\ of\ words_{t+1}$
    - Ergodic
    - Formula

With summing over all possible sequences.

$$\begin{aligned} H(L) &= \lim_{n\to\infty} \frac{1}{n} H(w_1, w_2, \ldots, w_n) \\ &= -\lim_{n\to\infty} \frac{1}{n} \sum_{W\in L} p(w_1, \ldots, w_n) \log p(w_1, \ldots, w_n) \end{aligned}$$

⋙

With a single sequence long enough

$$H(L) = \lim_{n\to\infty} -\frac{1}{n} \log p(w_1 w_2 \ldots w_n)$$

- Intuition of theorem:
  - A long-enough sequence of words contain many other shorter sequences.
  - Each of shorter sequences will reoccur in the longer sequence by their prob.

- Stationarity:
  - N-grams(stationary)
    - P_i is dependent only on P_{i-1} = P_{i+x} is dependent only on P_{i+x-1}
  - Natural Language(non-stationary)
    - P can be dependent on events(arbitrarily distant and time dependent).
    - The statistical models only give approximation to the correct distributions and entropies of NL.

Advanced: Perplexity's Relation to Entropy

- In Shannon-McMillan-Breiman theorem
  - we can compute entropy rate of stochastic process (L)
  - It is possible thanks to
    - The assumption that language is regular in certain ways(stationary and ergodic). ← "Incorrect"
    - The intuition(mentioned in the previous page)← "convenient simplifying"
- In fact, NL is not that way.
- A stochastic process only an approximation to the correct distributions and entropies of NL.
- Nevertheless, with incorrect but convenient simplifying assumption, we can compute the entropy of some stochastic process
  - by taking a very long sample of the output.
  - by computing its average log prob.

- Cross-entropy
  - Useful without knowing the actual prob distribution p.
  - Use some m(model of approximation to p).
  - Cross-entropy of m on p is.

Entropy (rate)

$$-\lim_{n\to\infty}\frac{1}{n}\sum_{W\in L} p(w_1,\ldots,w_n)\log p(w_1,\ldots,w_n)$$

Cross-Entropy (rate)

$$H(p,m)=\lim_{n\to\infty}-\frac{1}{n}\sum_{W\in L} p(w_1,\ldots,w_n)\log m(w_1,\ldots,w_n)$$

- ● Draw sequences according to p.
- ● Log of sequences' prob according to m.
- ∴ Predict p using m

- With Shannon-McMillan-Breiman theorem
  - Estimate cross-entropy of a model m on p by taking a single sequence long enough like entropy.

Entropy (rate)

$$H(p)=\lim_{n\to\infty}-\frac{1}{n}\log p(w_1w_2\ldots w_n)$$

Cross-Entropy (rate)

$$H(p,m)=\lim_{n\to\infty}-\frac{1}{n}\log m(w_1w_2\ldots w_n)$$

- What is useful…

$$H(p)\leq H(p,m)$$

Entropy (rate) **Cross-Entropy** (rate)

- For any model m, cross-entropy is an upper bound on the entropy
- Simplified model m helps estimate the true entropy of a sequence drawn according to p.
- $Difference_{H(p),\,H(p,m)}$ = a measure of how accurate a model is.
- Between model m_1, m_2, the more accurate model will be one the lower cross-entropy.

Advanced: Perplexity's Relation to Entropy

- Relation between perplexity and cross-entropy
  - Cross-entropy is in the limit as the length of the word sequence goes to infinity.
  - Need to Approximate to the cross-entropy of a n-gram model $M = P(w_i|w_{i-N+1:i-1})$ on a sequence of words W

n goes to infinity

$$\boxed{H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \dots w_n)}$$

⋙

Approximation w/ a sufficiently long sequence of fixed length

$$\boxed{H(W) = -\frac{1}{N}\log P(w_1 w_2 \dots w_N)}$$

- Perplexity of a model P is formally defined as 2 raised to the power of the cross-entropy.

(Cross-entropy) $log P(w_{1,}w_2 \dots w_N)^{-\frac{1}{N}}$

$$\begin{aligned}
\text{Perplexity}(W) &= 2^{H(W)} \\
&= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\
&= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}
\end{aligned}$$

Refer to if it is needed

$$2^{\log_2 P(w_{1,\dots,}w_n)^{-\frac{1}{N}}} = t$$

$$\log_2 2^{\log_2 P(w_{1,\dots,}w_n)^{-\frac{1}{N}}} = \log_2 t$$

$$\log_2 P(w_{1,\dots,}w_n)^{-\frac{1}{N}} = \log_2 t$$

$$\log_2 P(w_{1,\dots,}w_n)^{-\frac{1}{N}} * \log_2 2 = \log_2 t$$

$$P(w_{1,\dots,}w_n)^{-\frac{1}{N}} = t = 2^{H(w)}$$

End of Document