Unit 6

NGRAMS

N-Grams and Language models

- The simplest model that assigns probabilities to sentences and sequences of words, is known as **N-gram model.** It predicts the next word from the previous **N-1** words. (word prediction)

- Models that assign probabilities to sequences of words are called **language Models or LMs.**
- Computing the probability of the next word will turn out to be closely related to computing the probability of a sequence of words.
- The following sequence, for example, has a **non-zero probability** of appearing in a text:

  '. . . ***all of a sudden I notice three guys standing on the sidewalk...'***

  while this same set of words in a different order has a very low probability:

  '***on guys all I of notice sidewalk three a sudden standing the***'

- N-grams that assign a conditional probability to possible next words can be used to assign a joint probability to an entire sentence.
- N-gram model is one of the most important tools in speech and language processing.

- N-grams are essential in any task in which we have to identify words in noisy, ambiguous input.
  - In speech recognition, for example, the input speech sounds are very confusable and many words sound extremely similar.

Probabilistic Language Model-Applications

- Context Sensitive Spelling Correction

  Example: ***The office is about 15 minuets from my house.***

  Minuets means ***slow ballroom dance***

  P(minutes from my house)>P(minuets from my house)

- Natural language Generation

  Whenever you have to generate sentences again, look into which particular generation has the higher probability?

- Speech Recognition

  P(I saw a man)>>P(eyes awe of an)

Application of N-Grams

- **Augmentative communication-** (Newell et al., Communication 1998) systems that help the disabled.

- People who are unable to use speech or sign language to communicate, like the physicist Steven Hawking, can communicate by using simple body movements to select words from a menu that are spoken by the system.

- Word prediction can be used to suggest ***likely words for the menu***.

- Besides these sample areas, *N*-grams are also crucial in NLP tasks like **part-of-speech tagging**, **natural language generation**, and **word similarity**, as well as in applications from **authorship identification** and **sentiment extraction** to **predictive text input** systems for cell phones.

Probabilistic Language Model-Applications

- **Machine Translation**: The process of using artificial intelligence to automatically translate text from one language to another without human involvement.
- Modern machine translation goes beyond simple word-to-word translation to communicate the full meaning of the original language text in the target language.

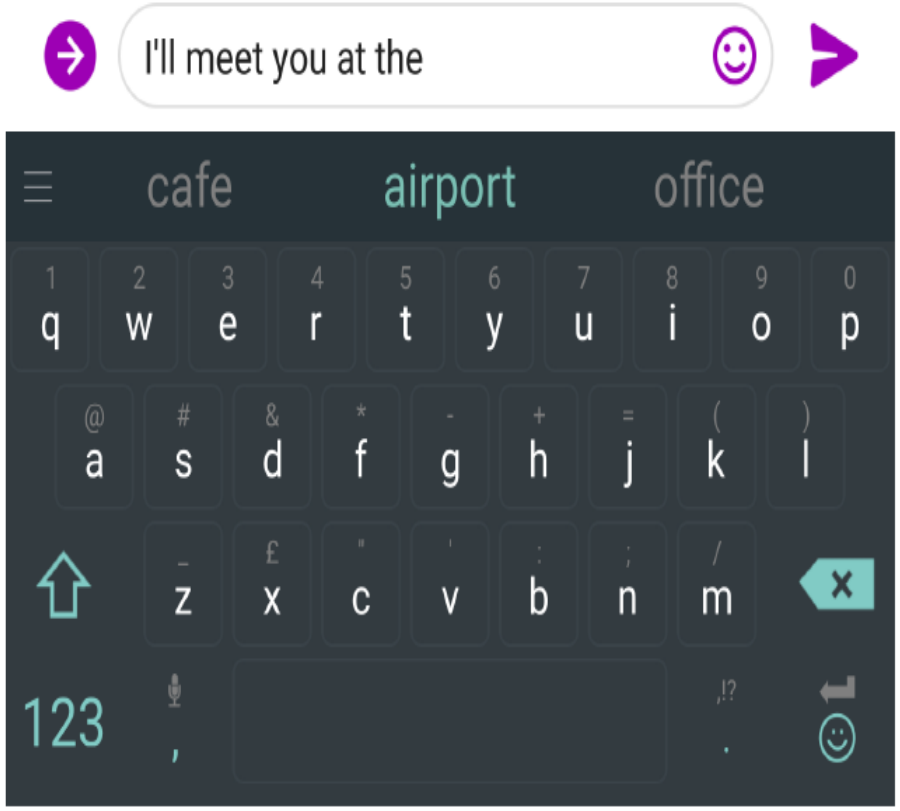
- It analyzes all text elements and recognizes how the words influence one another.
  - -problem of collocations

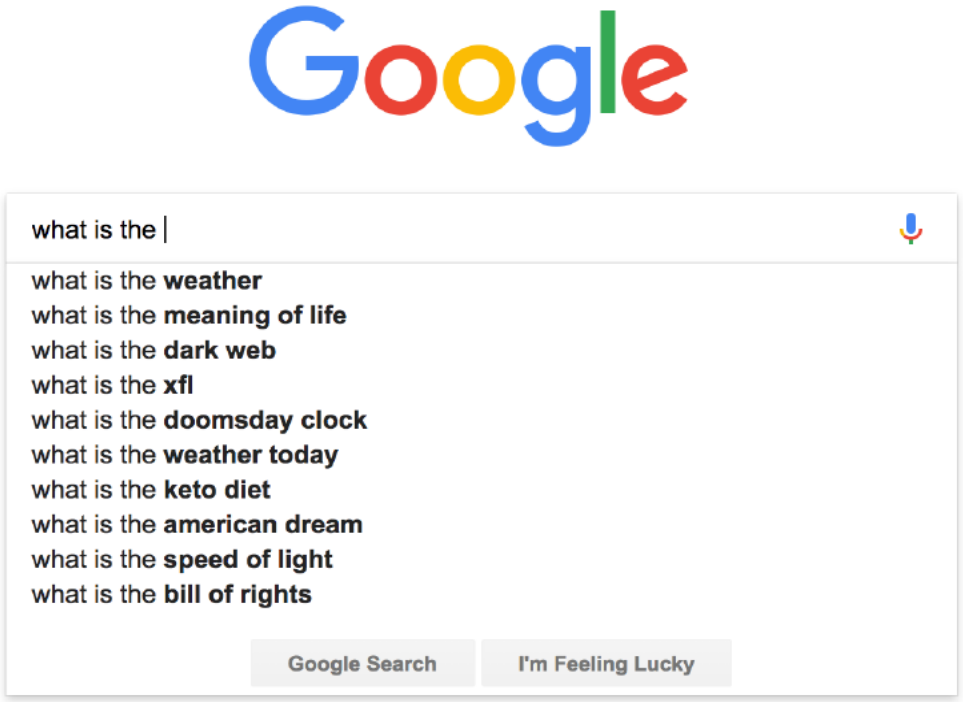
Example: P(high winds) >P(large winds)

P(I went to the movie) > (I flew to the movie)

- Completion Prediction
  - Language model also predicts the completion of a sentence
    - Please turn off your cell ……
    - Your program does not……
  - Predictive text input systems can guess what you are typing and give choices on how to complete it

Language Models Everyday





N-GRAMS

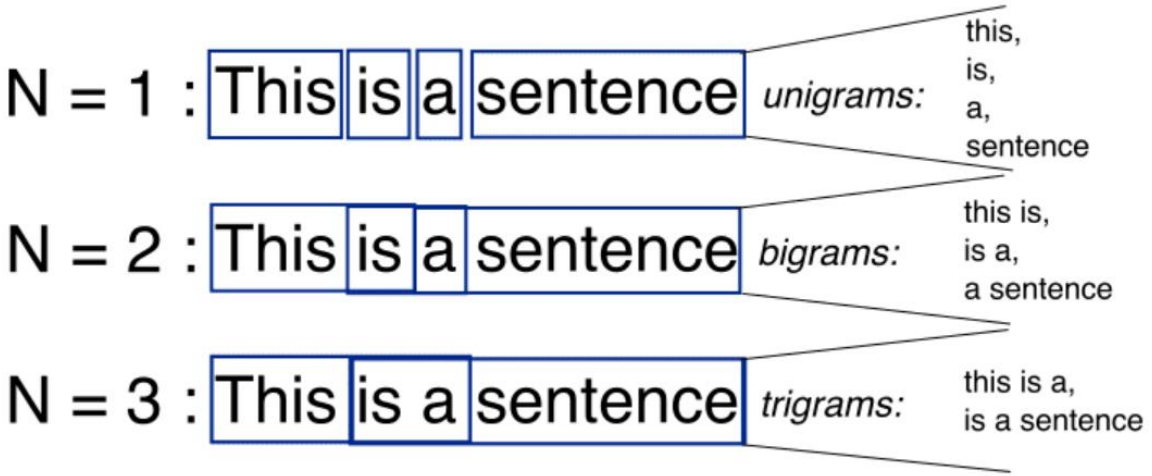


Figure 2 Uni-gram, Bi-gram, and Tri-gram Model

•An N-gram is a sequence of N words: **For eg: *Please turn your homework ….***

• A 2-gram (or bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”

•A 3-gram (or trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.

N-GRAMS: Counting Words in Corpus

- Probabilities are based on counting things.
- Counting of things in natural language is based on a corpus (plural corpora), an online collection of text CORPUS or speech.
- Two popular corpora:
  - **Brown**
  - **Switchboard**
- The **Brown corpus** is a 1-million-word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.), assembled at Brown University in 1963-64.

How many words are in the following Brown sentence?

Example: He stepped out into the hall, was delighted to encounter a water brother.

- Has 13 words if we don't count punctuation marks as words, 15 if we count punctuation. Whether we treat period (“.”), comma (“,”), and so on as words depends on the task

N-GRAMS-Counting Words in Corpus

- The **Switchboard corpus** of telephone conversations between strangers was collected about 3 million words in the early 1990s.

- Such corpora of spoken language don't have punctuation, but do introduce other complications with regard to defining words called **utterance** i.e., the spoken correlate of a sentence:

Example: I do uh main- mainly business data processing

- This **utterance** has two kinds of **disfluencies**.
  - The broken-off word main- is called a **fragment**.
  - Words like uh and um are called **fillers** or **filled pauses**.

SIMPLE (UNSMOOTHED) N-GRAMS

- **Goal**: To compute the probability of a **word w** given some **history h**, or **P(w|h)**.
- Suppose the history ***h*** is "***its water is so transparent that***" and we want to know the probability that the next word is ***the***: ***P(the|its water is so transparent that)***.
- One way is to estimate it from relative frequency counts.

How to estimate these probabilities

- One way is to estimate it from **relative frequency counts.**
- For example, we could take a very large corpus, count the number of times we see ***'its water is so transparent that'***, and count the number of times this is followed by ***the***.
- This would be answering the question "Out of the times we saw the history *h*, how many times was it followed by the word *w*",

$$\mathbf{P(}\textit{the}\mid\mathbf{its\ water\ is\ so\ transparent\ that).} = \frac{\mathbf{C(its\ water\ is\ so\ transparent\ that\ the)}}{\mathbf{C(its\ water\ is\ so\ transparent\ that)}}$$

Chain Rule of Probability

Disadvantage of Relative frequency count:

- This method of estimating probabilities depends upon **word counts** that work fine in many cases, but it won't be enough to give good estimates in most cases.
    - This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentences.
    - Even simple extensions of the example sentence may have counts of zero on the web (such as "***Walden Pond's water is so transparent that the***").
- The **joint probability** of an entire sequence of words needs lots of estimation.
- **Solution: Chain Rule of Probability**

Chain Rule of Probability

Let a sequence of **N** words either as $\boldsymbol{w_1 \ldots\ldots w_n}$ **or** $\boldsymbol{w_1^n}$.
For the joint probability of each word in a sequence having a particular value **P(X =** $\boldsymbol{w_1}$**;Y =** $\boldsymbol{w_2}$**;Z =** $\boldsymbol{w_3}$**;…..;W =** $\boldsymbol{w_n}$**)** we'll use **P(**$\boldsymbol{w_1; w_2; \ldots..; w_2}$**).**

$$P(X_1 \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1^2)\ldots P(X_n|X_1^{n-1})$$
$$= \prod_{k=1}^{n} P(X_k|X_1^{k-1})$$

Applying the chain rule to words, we get

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

The Chain Rule

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =
P(its) × P(water|its) × P(is|its water)
× P(so|its water is) × P(transparent|its water is so)

- The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.,

- But using the chain rule doesn’t really seem to help us! We don’t know any way to compute the exact probability of a word given a long sequence of preceding words,

- The intuition of the N-gram model is that instead of computing the probability of a word given its entire history, we will approximate the history by just the last few words.

The **bigram** model, for example, approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by using only the conditional probability of the preceding word $P(w_n|w_{n-1})$. In other words, instead of computing the probability

$$P(\text{the}|\text{Walden Pond's water is so transparent that})$$

we approximate it with the probability

$$P(\text{the}|\text{that})$$

When we use a bigram model to predict the conditional probability of the next word we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

Markov Assumption

- Markov Models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past

Markov Assumption

- Simplifying assumption:

Andrei Markov (1856~1922)

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Only previous k words are considered

Thus the general equation for this N-gram approximation to the conditional probability of the next word in a sequence is:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Markov Assumption

- P(the|its water is so transparent that)
- An N-gram model uses only N-1 words of prior context
  - Unigram: P(the)
  - Bigram: P(the|that)
  - Trigram: P(the|transparent that)

N-Gram Model

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
  - because language has **long-distance dependencies**:

"The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing."

- But we can often get away with N-gram models

Estimating Bigram Probabilities

- An intuitive way to estimate probabilities is called **maximum likelihood estimation or MLE**
- We get maximum likelihood estimation the MLE estimate for the parameters of an **N-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1**
- For example, to compute a particular bigram probability of a word y given a previous word x, we'll compute the count of the bigram C(xy) and normalize by the sum of all the bigrams that share the same first word x

Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)
  - relative frequency based on the empirical counts on a training set

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

c – count

Estimating Bigram Probabilities

An example

$$P(w_i \mid w_{i-1}) \overset{\text{MLE}}{=} \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$
$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

Estimating Bigram Probabilities

	<s>	I	AM	SAM	</s>
<s>	0	0.67	0	0	0
I	0	0	0.67	0	0
AM	0	0	0	0.5	0
SAM	0	0.5	0	0	0.5
</s>	0	0	0	0	0

P(<s> I AM SAM<s>)=P(I|<s>)*P(AM|I)*P(SAM|AM)*P(</s>|SAM)

=0.67*0.67*0.5*0.5=0.1122

Example:

- Unigram Model

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

Example:

$$P(the, prime, minister, of, our, country)$$

$$= P(the) * P(prime) * P(minister) * P(of) * P(our)$$

$$* P(country)$$

- Bigram Model:

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

Example:

$$P(the, prime, minister, of, our, country)$$

$$= P(the) * P(prime \mid the) * P(minister \mid prime)$$

$$* P(of \mid minister) * P(our \mid of) * P(country \mid our)$$

- Trigram Model

$$P(w_1, w_2, \ldots, w_n)$$

$$= P(w_1) * P(w_2 \mid w_1) * P(w_3 \mid w_1, w_2) * P(w_4 \mid w_2, w_3) * \ldots$$

$$* P(w_n \mid w_{n-2}, w_{n-1})$$

Example:

$$P(the, prime, minister, of, our, country)$$

$$= P(the) * P(prime \mid the) * P(minister \mid the, prime)$$

$$* P(of \mid prime, minister) * P(our \mid minister, of)$$

$$* P(country \mid of, our)$$

A bigger example: Berkeley Restaurant Project sentences

- Berkeley Restaurant Project, a dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California (Jurafsky et al., 1994). X
- Here are some sample user queries, lowercased and with no punctuation (a representative corpus of 9332 sentences is on the website):
- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast

A bigger example: Berkeley Restaurant Project sentences

	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 4.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

A bigger example: Berkeley Restaurant Project sentences

Raw bigram probabilities

$$P(w_i \mid w_{i-1}) \overset{\text{MLE}}{=} \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	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

A bigger example: Berkeley Restaurant Project sentences

Here are a few other useful probabilities:

$P(\texttt{i}\,|\,\texttt{<s>}) = 0.25$ $\quad$ $P(\texttt{english}\,|\,\texttt{want}) = 0.0011$

$P(\texttt{food}\,|\,\texttt{english}) = 0.5$ $\quad$ $P(\texttt{</s>}\,|\,\texttt{food}) = 0.68$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

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

A bigger example: Berkeley Restaurant Project sentences

- Compute the probability of **I want chinese food**
- P(<s> I want chinese food </s>)
- =P(I|<s>) P(want|I)P(chinese|want)P(food|chinese)P(</s>|food)
- =0.25x0.33x0.0065x0.52x0.68
- =0.00018
- These probabilities get super tiny when we have longer inputs more infrequent words

- **Training corpus:**
  - ***<s> I am from Manipal </s>***
  - ***<s> I am a teacher </s>***
  - ***<s> students are good and are from various cities</s>***
  - ***<s> students from Manipal do engineering</s>***
- **Test data:**
  - ***<s> students are from Manipal </s>***

- As per the Bigram model, the test sentence can be expanded as follows to estimate the bigram probability;

P(<s> students are from Manipal </s>)

= P(students | <s>) * P(are | students) * P(from | are)
**** P(Manipal | from) * P(</s> | Manipal)***

$$P(w_n \mid w_{n-1}) = \frac{count(w_{n-1}, w_n)}{count(w_{n-1})}$$

$$P(students \mid < s >) = \frac{count(< s > students)}{count(< s >)} = \frac{2}{4} = \frac{1}{2}$$

$$P(are \mid students) = \frac{count(students\ are)}{count(students)} = \frac{1}{2}$$

$$P(from \mid are) = \frac{count(are\ from)}{count(are)} = \frac{1}{2}$$

$$P(Manipal|from) = \frac{count(from\ Manipal)}{count(from)} = \frac{2}{3}$$

$$P(</s > |Manipal) = \frac{count(Manipal\ </s >)}{count(Manipal)} = \frac{1}{2}$$

P(<s> students are from Manipal</s>) ***= 1/2 * 1/2 * 1/2 * 2/3 * 1/2 = 0.0416***

Method-2

- **Unigram count matrix**

<s>	students	are	from	Manipal	</s>
4	2	2	3	2	4

- **Bigram count matrix**

		w_n				
		students	are	from	Manipal	</s>
w_{n-1}	<s>	2	0	0	0	0
	students	0	1	1	0	0
	are	0	0	1	0	0
	from	0	0	0	2	0
	Manipal	0	0	0	0	1

- **Bigram probability matrix (normalized by unigram counts)**

		w_n				
		students	are	from	Manipal	</s>
w_{n-1}	<s>	2/4	0/4	0/4	0/4	0/4
	students	0/2	1/2	1/2	0/2	0/2
	are	0/2	0/2	1/2	0/2	0/2
	from	0/3	0/3	0/3	2/3	0/3
	Manipal	0/2	0/2	0/2	0/2	1/2

P(<s> students are from Manipal</s>)

*= 1/2 * 1/2 * 1/2 * 2/3 * 1/2 = 0.0416*

- The training-and-testing paradigm can be used to evaluate different N-gram architectures.
- To compare different language models , take a corpus and divide it into a **training set and a test set**.
- Then we train the two different N-gram models on the training set and see which one better models the test set.
- There is is a useful metric for how well a given statistical model matches a test corpus, called **perplexity.**
- **Perplexity** is based on computing the probability of each sentence in the test set; intuitively, whichever model assigns a higher probability to the test set (hence more accurately predicts the test set) is a better model.
- Since the evaluation metric is based on test set probability, it's important not to let the test sentences into the training set.
- Suppose we are trying to compute the probability of a particular "test" sentence. If the test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. This situation is called **training on the test set**.
- Training on the test set introduces a bias that makes the probabilities all look too high and causes huge inaccuracies in perplexity.

- In addition to training and test sets, other divisions of data are often useful.
- Sometimes we need an extra source of data to augment the training set. Such extra data is called a **held-out** set, because we hold it out from our training set when we train our N-gram counts. The held-out corpus is then used to set some other parameters.
- Finally, sometimes we need to have multiple test sets.
- This happens because we might use a particular test set so often that we implicitly tune to its characteristics. Then we would definitely need a fresh test set which is truly unseen. In such cases, we call the initial test set the **development test set or, devset.**

EVALUATING N-GRAMS: PERPLEXITY

- The best way to evaluate the performance of a language model is to embed it in an application and measure the total performance of the application.
- Such end-to-end evaluation is called **extrinsic** evaluation, and also sometimes called in **vivo evaluation** (Sparck Jones and Galliers, 1996).
- Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand.
- Thus for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.
- Unfortunately, end-to-end evaluation is often very expensive; evaluating a large speech recognition test set, for example, takes hours or even days.
- Thus we would like a metric that can be used to quickly evaluate potential improvements in a language model.
- **An intrinsic evaluation** metric is one which measures the quality of a model independent of any application.
- **Perplexity is the most common intrinsic evaluation metric** for N-gram language models. While an (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in speech recognition performance (or any other end-to-end metric), it often correlates with such improvements.
- Thus it is commonly used as a quick check on an algorithm and an improvement in perplexity can then be confirmed by an end-to-end evaluation.

- The intuition of perplexity is that given two probabilistic models, the better model is the one that has a tighter fit to the test data, or predicts the details of the test data better.
- We can measure better prediction by looking at the probability the model assigns to the test data; the better model will assign a higher probability to the test data.
- More formally, the perplexity (sometimes called PP for short) of a language model on a test set is a function of the probability that the language model assigns to that test set.
- For a test set W = w1w2 ...wN, the perplexity is the probability of the test set, normalized by the number of words:

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

We can use the chain rule to expand the probability of W:

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

Thus if we are computing the perplexity of W with a bigram language model, we get:

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

- Because of the inverse in equation, the higher the conditional probability of the word sequence, the lower the perplexity.
- Thus minimizing perplexity is equivalent to maximizing the test set probability according to the language model.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

- What we generally use for word sequence is the entire sequence of words in some test set.
- Since this sequence will cross many sentence boundaries**, we need to include the begin- and end-sentence markers <s> and </s> in the probability computation.**
- **We also need to include the end-of-sentence marker </s> (but not the beginning-of-sentence marker <s>)**) in the total count of word tokens N.

Practical Issues

- There is a major problem with the maximum likelihood estimation process we have seen for training the parameters of an N-gram model.
- This is the problem of sparse data caused by the fact that our maximum likelihood estimate was based on a particular set of training data.
- For any N-gram that occurred a sufficient number of times, we might have a good estimate of its probability.
- But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.
- This missing data means that the N-gram matrix for any given training corpus is bound to have a very large number of cases of putative “zero probability N-grams” that should really have some non-zero probability.
- Furthermore, the MLE method also produces poor estimates when the counts are non-zero but still small.
- We need a method which can help get better estimates for these zero or low frequency counts.
- Zero counts turn out to cause another huge problem. The perplexity metric defined above requires that we compute the probability of each test sentence.
- But if a test sentence has an N-gram that never appeared in the training set, the Maximum Likelihood estimate of the probability for this N-gram, and hence for the whole test sentence, will be zero!
- This means that in order to evaluate our language models, we need to modify the MLE method to assign some non-zero probability to any N-gram, even one that was never observed in training.
- For these reasons, we’ll want to modify the maximum likelihood estimates for computing N-gram probabilities, focusing on the N-gram events that we incorrectly assumed had zero probability.
- We use the term smoothing for such modifications that address the poor estimates that are due to variability in small data sets.

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

How do you handle unseen n-grams?

- **Smoothing**

Use some of the probability mass to cover unseen events

- **Backoff**

Use counts from a smaller context

- **Interpolation**

Combine multiple sources of information appropriately weighted

Smoothing

- Since there are a combinatorial number of possible word sequences, many rare (but not impossible) combinations never occur in training, so MLE incorrectly assigns zero to many parameters (***sparse data***).

- If a new combination occurs during testing, it is given a probability of zero and the entire sequence gets a probability of zero (i.e. infinite perplexity).

- Modify the maximum likelihood estimates for computing N-gram probabilities, focusing on the N-gram events that we incorrectly assumed had zero probability.
- We use the term smoothing for such modifications that address the poor estimates that are due to variability in small data sets
- The name comes from the fact that(looking ahead a bit) we **will be saving a little bit of probability mass from the higher counts, and piling it instead on the zero counts**, making the distribution a little less jagged.

Laplace Smoothing

- One simple way to do smoothing might be just to take our matrix of bigram counts, before we normalize them into probabilities, and add one to all the counts. This algorithm is called Laplace smoothing, or Laplace's Law
- Laplace smoothing does not perform well enough to be used in modern N-gram models, but it introduces many of the concepts that is used in other smoothing algorithms and also gives us a useful baseline
- **Laplace smoothing to unigram probabilities:**
  - Recall that the unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N:

$$P(w_i) = \frac{c_i}{N}$$

Laplace Smoothing

- Laplace smoothing merely adds one to each count (hence its alternate name **add one smoothing**)

- Since there are V words in the vocabulary, and each one got incremented, we also need to adjust the denominator to take into account the extra V observations

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

Laplace Smoothing

- Instead of changing both the numerator and denominator it is convenient to describe how a smoothing algorithm affects the numerator, by defining an **adjusted count c***
- This adjusted count is easier to compare directly with the MLE counts, and can be turned into a probability like an MLE count by normalizing by N
- To define this count, since we are only changing the numerator, in addition to adding one we'll also need to multiply by a normalization factor $\frac{N}{N+V}$:

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

We can now turn c_i* into a probability P_i* by normalizing by N.

Laplace Smoothing

- A related way to view smoothing is as **discounting (lowering)** some non-zero counts in order to get the probability mass that will be assigned to the zero counts.
- Thus, instead of referring to the discounted counts c*, we might describe a smoothing algorithm in terms of a relative discount d_c, the ratio of the discounted counts to the original counts:

$$d_c = \frac{c^*}{c}$$

Unigram Smoothing Example

- Tiny Corpus, V=4; N=20

$$P_{LP}(w_i) = \frac{c_i + 1}{N + V}$$

Word	True Ct	Unigram Prob	New Ct	Adjusted Prob
eat	10	.5	11	.46
British	4	.2	5	.21
food	6	.3	7	.29
happily	0	.0	1	.04
	20	1.0	~20	1.0

11/(20+4)

5/(20+4)

Example: Berkeley Restaurant

	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

Fig 1:Unsmoothed Bigram 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

Fig 2:-Add-one smoothed bigram counts for eight of the words (out of V = 1446) in the Berkeley Restaurant Project corpus of 9332 sentences.

Example: Berkeley Restaurant

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	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

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

For add-one smoothed bigram counts we need to augment the unigram count by the number of total word types in the vocabulary V:

Example: Berkeley Restaurant

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	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

Add-one smoothed bigram probabilities for eight of the words (out of V = 1446) in the BeRP corpus of 9332 sentences

Reconstituted counts

- It is often convenient to reconstruct the count matrix, to see how much a smoothing algorithm changed the original counts.
- The adjusted counts are computed as :

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

	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

Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences.

1 x 2533/(2533+1446)

609 x 927/(927+1446)

Example: Berkeley Restaurant
(Big Change to the Counts!)

- Note that add-one smoothing has made a very big change to the counts **C(want to) changed from 608 to 238!**
- We can see this in probability space as well: P(to|want) decreases from .66 in the unsmoothed case to .26 in the smoothed case

- Looking at the **discount d** (the ratio between new and old counts) shows us how strikingly the counts for each prefix word have been reduced; the discount for the bigram **want to is .39**, while the discount for Chinese food is **.10,** a factor of 10!

- The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros
- *We could move a bit less mass by adding a fractional count rather than 1 (add-δ smoothing; (Lidstone, 1920; Johnson, 1932; Jeffreys, 1948)), but this method requires a method for choosing δ dynamically, results in an inappropriate discount for many counts, and turns out to give counts with poor variance.*

Example: Berkeley Restaurant

Compare with raw bigram counts

	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

	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

Problem with Laplace Smoothing

- Problem: gives too much probability mass to unseen n-grams.
- For sparse sets of data over large vocabularies, such as n-grams, Laplace's law actually gives far too much of the probability space to unseen events.

Good-Turing Smoothing

- Also called Good-Turing discounting, Good-Turing estimation
- Intuition is to use the count of things we've seen once to help estimate the count of things we've never seen.
- The Good-Turing algorithm was first described by Good (1953), who credits Turing with the original idea.
- The basic insight of Good-Turing smoothing is to re-estimate the amount of probability mass to assign to N-grams with zero counts by **looking at the number of N-grams that occurred one time.**
- A word or N-gram (or any event) that occurs once is called a **singleton, or a hapax legomenon.**
- The Good-Turing intuition is to use the frequency of singletons as a re-estimate of the frequency of zero-count bigrams

Good-Turing Smoothing

- The Good-Turing algorithm is based on computing Nc, the number of N-grams that occur c times.
- We refer to the number of N-grams that occur c times as the **frequency of frequency c**.
- So applying the idea to smoothing the joint probability of bigrams, N0 is the number of bigrams with count 0, N1 the number of bigrams with count 1 (singletons), and so on.
- We can think of each of the **Nc as a bin which stores the number of different N-grams that occur in the training set with that frequency c.**
- Formally,

$$N_c = \sum_{x:count(x)=c} 1$$

- The MLE count for Nc is c. The Good-Turing estimate replaces this with a smoothed count c $*$, as a function of Nc+1:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

Good-Turing Smoothing

- The above equation can be used to replace the MLE counts for all the bins N1, N2, and so on.
- Instead of using this equation directly to re-estimate the smoothed count c $*$ for N0, we use the following equation for the probability P $*$ GT for things that had zero count N0, or what we might call the **missing mass.**

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

- Here N1 is the count of items in bin 1, i.e. that were seen once in training, and N is the total number of items we have seen in training.

Good-Turing Smoothing

- The Good-Turing method was first proposed for estimating the populations of animal species.
- Consider an illustrative example from this domain created by Joshua Goodman and Stanley Chen.
  S
- Imagine you are fishing

  There are 8 species in the lake: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You catch:

  10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

What is the probability next fish caught is from a new species (one not seen in our previous catch, ie. The one has 0 count in training set)?

And how likely is it that the next species is another trout?

	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{GT}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{GT}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Good-Turing Smoothing

- The revised count c ∗ for trout was discounted from c = 1.0 to c ∗ = .67, (thus leaving some probability mass p ∗ GT(unseen) = 3 /18 = .17 for the catfish and bass).
- And since we know there were 2 unknown species, the probability of the next fish being specifically a catfish is p ∗ GT(catfish) = 1 /2 × 3 /18 = .085.

AP Newswire			Berkeley Restaurant		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Bigram "frequencies of frequencies" and Good-Turing re-estimations for the 22 million AP bigrams from Church and Gale (1991) and from the Berkeley Restaurant corpus of 9332 sentences.

Backoff And Interpolation

- The discounting we have been discussing so far can help solve the problem of zero frequency N-grams
- But there is an additional source of knowledge we can draw on
- If we are trying to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$.
- Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.

Backoff And Interpolation

- In other words, sometimes using less context is a good thing, helping to generalize more for contexts that the model hasn't learned much about
- There are two ways backoff to use this N-gram "hierarchy"
- In backoff, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram
- In other words, we only "back off" to a lower-order N-gram if we have zero evidence for a higher-order interpolation N-gram
- By contrast, in interpolation, we always mix the probability estimates from all the N-gram estimators, weighing and combining the trigram, bigram, and unigram counts.

Backoff And Interpolation

- In simple linear interpolation, we combine different order N-grams by linearly interpolating all the models
- Thus, we estimate the trigram probability $P(w_n|w_{n-2}w_{n-1})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted b: