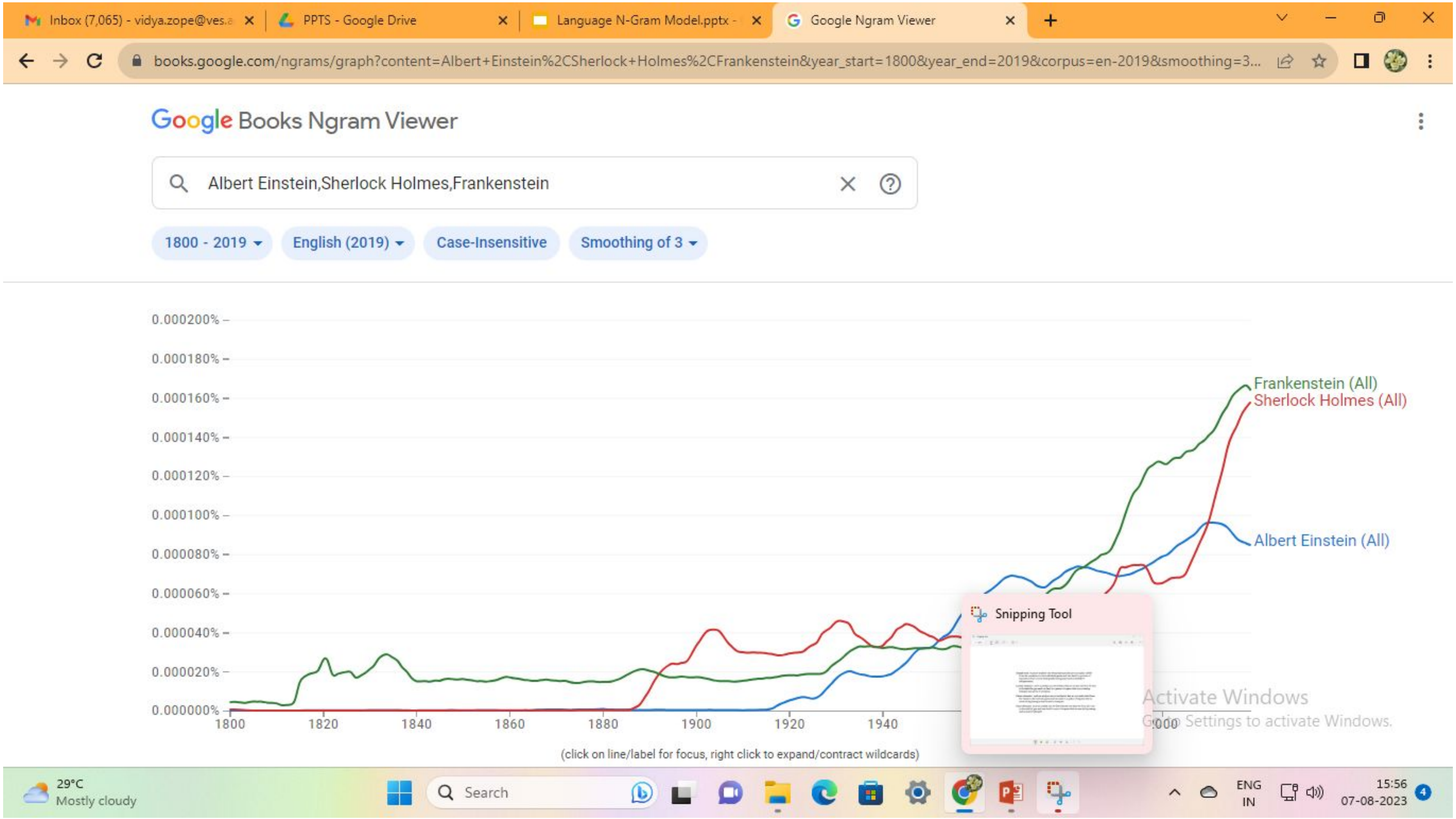
Word Level Analysis

Language N-Gram Model– Module II

https://books.google.com/ngrams




```
print ngrams("I am eating pizza.", n=2) # bigrams

[('I', 'am'), ('am', 'eating'), ('eating', 'pizza')]
```

Probabilistic Language Models

- Today's goal: assign a probability to a sentence

Why?

- Machine Translation:
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - The office is about fifteen **minuets** from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- + Summarization, question-answering, etc., etc.!!

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4)$$

- A model that computes either of these:

$P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$ is called a **language model**.

- Better: **the grammar** But **language model** or **LM** is standard

What is a language model?

- Probability distributions over sentences (i.e., word sequences) : $\mathbf{P(W) = P(w_1 w_2 w_3 w_4 \ldots w_k)}$
- Can use them to generate strings
- $\mathbf{P(w_k \mid w_2 w_3 w_4 \ldots w_{k-1})}$
- Rank possible sentences
  - P("Today is Thursday") > P("Thursday Today is")
  - P("Today is Thursday") > P("Today is Sunny")

Uses of Language Models

- Speech recognition
  - “I ate a cherry” is a more likely sentence than “Eye eight uh Jerry”
- OCR & Handwriting recognition
  - More probable sentences are more likely correct readings.
- Machine translation
  - More likely sentences are probably better translations.
- Generation
  - More likely sentences are probably better NL generations.
- Context sensitive spelling correction
  - “Their are problems wit this sentence.”

Language model applications

- Context-sensitive spelling correction

Real Word Spelling Errors

- Words that sound the same
  - Their/they're/there
  - To/too/two
  - Weather/whether
  - Peace/piece
  - You're/your
- Typos that result in real words
  - Lave for Have

Language model applications

Autocomplete

Language model applications

- **Language generation : https://pdos.csail.mit.edu/archive/scigen/**

Deploying Superblocks and Compilers

Julia and Dan

Abstract

Recent advances in replicated algorithms and relational symmetries have paved the way for architecture. After years of natural research into erasure coding, we show the deployment of courseware, which embodies the key principles of steganography. *Loy*, our new system for the exploration of sensor networks, is the solution to all of these issues.

1 Introduction

Steganographers agree that robust symmetries are an interesting new topic in the field of cryptography, and information theorists concur. We view operat-

thesize unstable algorithms, we fulfi
without investigating the evaluation o

Our contributions are threefold. Fir
how erasure coding can be applied to
tion of reinforcement learning. We p
algorithm for the deployment of extr
ming (*Loy*), which we use to prove tha
and operating systems [19, 7, 14] can
fill this goal. we examine how replic
be applied to the deployment of linke

The rest of this paper is organized a
marily, we motivate the need for fibe
We demonstrate the synthesis of the T
Finally, we conclude.

Completion Prediction

- A language model also supports predicting the completion of a sentence.
  - Please turn off your cell ______
  - Kindly submit _______
- *Predictive text input* systems can guess what you are typing and give choices on how to complete it.

N-Gram Models of Language

- Use the previous N-1 words in a sequence to predict the next word
- Language Model (LM)
  - unigrams, bigrams, trigrams,…
- How do we train these models?
  - Very large corpora

Corpora

- **Corpora are online collections of text and speech**
  - Brown Corpus
  - Wall Street Journal
  - AP newswire
  - Hansards
  - Timit
  - DARPA/NIST text/speech corpora (Call Home, Call Friend, ATIS, Switchboard, Broadcast News, Broadcast Conversation, TDT, Communicator)
  - TRAINS, Boston Radio News Corpus

Terminology

- Sentence: unit of written language
- Utterance: unit of spoken language
- Word Form: the inflected form as it actually appears in the corpus
- Lemma: an abstract form, shared by word forms having the same stem, part of speech, word sense – stands for the class of words with same stem
- Types: number of distinct words in a corpus (vocabulary size)
- Tokens: total number of words

Different approaches

- **Rule-Based Models**: Simple, interpretable, but not scalable.
- **Statistical Language Models:** Effective for many tasks, but limited by context size.
- **Neural Language Models:** (learn representations of words and their contexts) Powerful and flexible, but computationally expensive.
- **Transformer-Based Models:**(self-attention mechanisms to handle dependencies regardless of distance) State-of-the-art performance, but require significant resources.
- **Lattice-Based Models:** Handle ambiguity well, but are complex and resource-intensive.
- **Hybrid Models:** Combine strengths of various models, but are complex to design and implement.

Lattice based language model

- A lattice in the context of inclusion relations is a partially ordered set (poset) in which any two elements have a unique least upper bound (supremum) and a greatest lower bound (infimum).
- This structure is used to model hierarchical relationships where elements can be combined or intersected in a well-defined manner.

Lattice-based language models

Translate the sentence "he likes me" into Marathi.

Possible Translations:

1. First, we need to identify the possible translations of each word in the sentence "he likes me" into Marathi. The translations could be:

He: "तो" (to)

Likes: "आवडतो" (avadto), "प्रिय" (priya), "पसंत करतो" (pasant karto)

Me: "मला" (mala)

Lattice-based language models

2. Constructing the Lattice:

```
(start)
   |
  (तो)
   |
(आवडतो)---(मला)
   |          |
(प्रिय)--------|
   |          |
(पसंत करतो)---|
   |
  (end)
```

3. Probabilities and Context: each edge would also have an associated probability, indicating the likelihood of that particular translation.

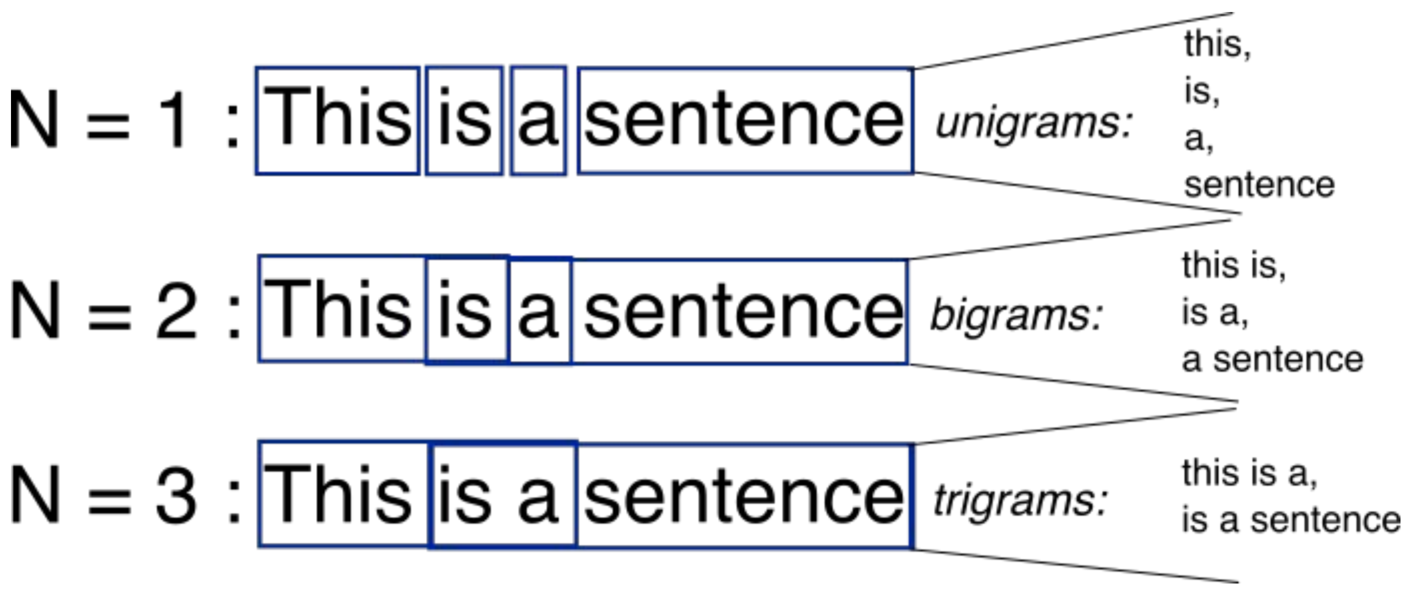
4. Choosing the Best Path:
तो आवडतो मला" (to avadto mala)
"तो प्रिय मला" (to priya mala)
"तो पसंत करतो मला" (to pasant karto mala)

5. Evaluation:
The correct context would likely give higher probability to the path that makes the most grammatical and contextual sense.

```
(start)
  |
(तो:1.0)
  |
(आवडतो:0.6) ----> (मला:1.0) ----> (end)
  |                   |
(प्रिय:0.2) ---------------|
  |                   |
(पसंत करतो:0.2) ----------|
```

Bag-of-Words with N-grams

- N-grams: a contiguous sequence of n tokens from a given piece of text



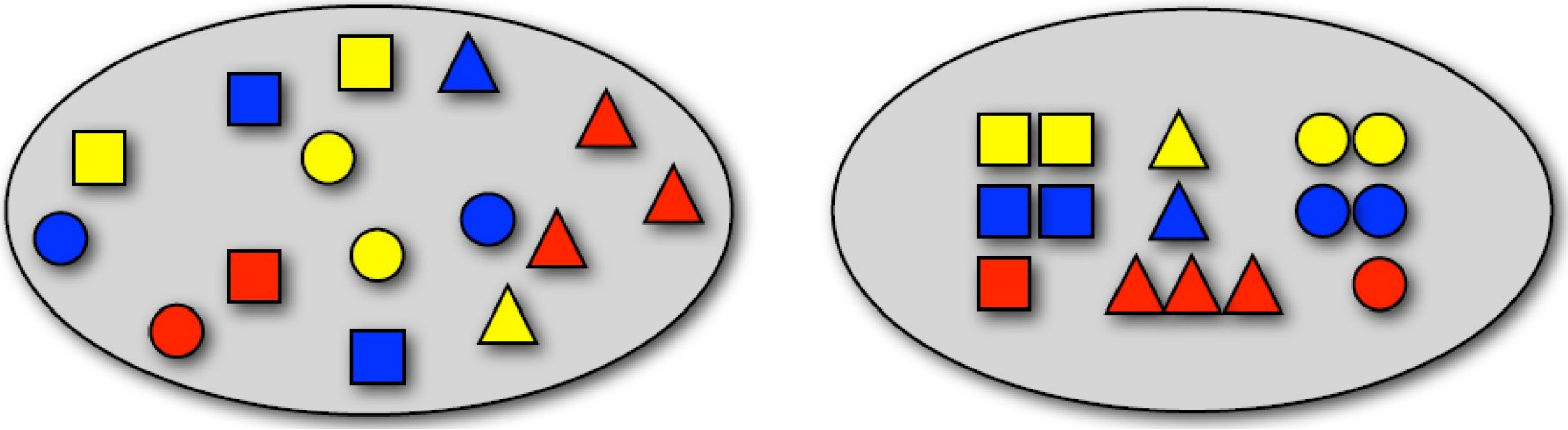
N-Gram Models

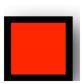
- **Unigram model:** $P(w_1)P(w_2)P(w_3)\ldots P(w_n)$
- **Bigram model:** $P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w_{n-1})$
- **Trigram model:**
- $P(w_1)P(w_2|w_1)P(w_3|w_2,w_1)\ldots P(w_n|w_{n-1}w_{n-2})$
- **N-gram model:**
- $P(w_1)P(w_2|w_1)\ldots P(w_n|w_{n-1}w_{n-2}\ldots w_{n-N})$

Random language via n-gram

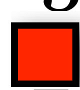
- http://www.cs.jhu.edu/~jason/465/PowerPoint/lect01,3tr-ngram-gen.pdf

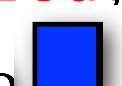
Sampling with replacement

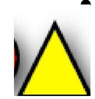




1. P( ) = ? 2. P(□) = ? 3. P(red, □) = ?







4. P(blue) = ? 5. P(red | □) = ?

6. P(□ | red) = ? 7. P( ) = ?

Sampling words with replacement

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

P(of) = 3/66	P(her) = 2/66
P(Alice) = 2/66	P(sister) = 2/66
P(was) = 2/66	P(,) = 4/66
P(to) = 2/66	P(') = 4/66

Example from Julia hockenmaier

How to compute P(W) ?

- How to compute this joint probability:
    - P(its, water, is, so, transparent, that)
- Intuition: let’s rely on the Chain Rule of Probability

Reminder : Chain Rule

- Recall the definition of conditional probabilities

Rewriting:

- More variables:

  $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$

- The Chain Rule in General

$$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$$

Reminder : Chain Rule

- The big red dog
- P(The)*P(big|the)*P(red|the big)*P(dog|the big red)
- Better P(The| <Beginning of sentence>) written as
  P(The | <S>)

Reminder : 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)

How to estimate these Probabilities ?

- Could we just count and divide?

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

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption

Andrei Markov

- Simplifying assumption:

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

- Or maybe

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

Language model with N-gram

- The chain rule:
- $\mathbf{P}(\boldsymbol{X_1}, \boldsymbol{X_2}, \ldots \boldsymbol{X_n}) = \mathbf{P}(\boldsymbol{X_1})\mathbf{P}(\boldsymbol{X_2}|\boldsymbol{X_1})\mathbf{P}(\boldsymbol{X_3}|\boldsymbol{X_2}, \boldsymbol{X_1}) \ldots \boldsymbol{P}(\boldsymbol{X_n} \mid \boldsymbol{X_1}, \ldots \boldsymbol{X_{n-1}})$
- N-gram language model assumes each word depends only on the last n-1 words (Markov assumption)

Language model with N-gram

- Example: trigram (3-gram)

$P(w_n \mid w_1, \ldots w_{n-1}) = P(w_n \mid w_{n-2}, w_{n-1})$

$P(w_1, \ldots w_n)=$

$P(w_1)P(w_2|w_1) \ldots P(w_n \mid w_{n-2}, w_{n-1})$

- **P("*Today is a sunny day*")**

=P("Today")P("is" |"Today")P("a" |"is", "Today")…

P("day" |"sunny", "a")

N-grams : Example – The big red dog

- Unigrams: P(dog)
- Bigrams: P(dog|red)
- Trigrams: P(dog|big red)
- Four-grams: P(dog|the big red)

In general, we'll be dealing with
P(Word| Some fixed prefix)

Simple N-Grams

- Assume a language has T word types in its lexicon, how likely is word x to follow word y?
  - Simplest model of word probability: 1/T
  - Alternative 1: estimate likelihood of x occurring in new text based on its general frequency of occurrence estimated from a corpus (unigram probability)

    popcorn is more likely to occur than unicorn

  - Alternative 2: condition the likelihood of x occurring in the context of previous words (bigrams, trigrams,…)

    mythical unicorn is more likely than mythical popcorn

An example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

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

bigrams in our tongue twister

Peter Piper picked a peck of pickled pepper.
Where's the pickled pepper that Peter Piper picked?

the conditional probability of "Piper" given "Peter":

$$) \ p(\text{Piper}|\text{Peter}) = \frac{|\text{Peter Piper}|}{|\text{Peter}|} = \frac{2}{2} = 1$$

$$p(\text{Piper}|\text{a}) = \frac{|\text{a Piper}|}{|\text{a}|} = \frac{0}{1} = 0$$

Bigrams	Bigram frequencies
picked a	1
pepper that	1
peck of	1
a peck	1
pickled pepper	2
Where s	1
Piper picked	2
the pickled	1
Peter Piper	2
of pickled	1
pepper Where	1
that Peter	1
s the	1

<s> Peter Piper picked a peck of pickled pepper. </s>
<s> Where's the pickled pepper that Peter Piper picked? </s>

$$
\begin{aligned}
& p\left(\begin{array}{l}\text{Where's the pickled pepper that Peter} \\ \text{Piper picked a peck of pickled pepper}\end{array}\right) \\
= \; & p(\text{Where's}|\text{<s>}) \times p(\text{the}|\text{Where's}) \times p(\text{pickled}|\text{the}) \times \\
& p(\text{pepper}|\text{pickled}) \times p(\text{that}|\text{pepper}) \times p(\text{Peter}|\text{that}) \times \\
& p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times p(\text{a}|\text{picked}) \times \\
& p(\text{peck}|\text{a}) \times p(\text{of}|\text{peck}) \times p(\text{pickled}|\text{of}) \times \\
& p(\text{peper}|\text{pickled}) \times p(\text{</s>}|\text{pepper}) \\
= \; & .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times 1 \times .5 \\
= \; & .0625
\end{aligned}
$$

A Simple Bigram Example

- Estimate the likelihood of the sentence I want to eat Chinese food.
  - P(I want to eat Chinese food) = P(I | <start>) P(want | I) P(to | want) P(eat | to) P(Chinese | eat) P(food | Chinese) P(<end>|food)
- What do we need to calculate these likelihoods?
  - Bigram probabilities for each word pair sequence in the sentence
  - Calculated from a large corpus

Early Bigram Probabilities from BERP

Eat on	.16	Eat Thai	.03
Eat some	.06	Eat breakfast	.03
Eat lunch	.06	Eat in	.02
Eat dinner	.05	Eat Chinese	.02
Eat at	.04	Eat Mexican	.02
Eat a	.04	Eat tomorrow	.01
Eat Indian	.04	Eat dessert	.007
Eat today	.03	Eat British	.001

<start> I	.25	Want some	.04
<start> I'd	.06	Want Thai	.01
<start> Tell	.04	To eat	.26
<start> I'm	.02	To have	.14
I want	.32	To spend	.09
I would	.29	To be	.02
I don't	.08	British food	.60
I have	.04	British restaurant	.15
Want to	.65	British cuisine	.01
Want a	.05	British lunch	.01

- P(I want to eat British food) = P(I|<start>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British) = .25*.32*.65*.26*.001*.60 = .000080
  - Suppose P(<end>|food) = .2?
  - How would we calculate I want to eat Chinese food ?
- Probabilities roughly capture ``syntactic'' facts and ``world knowledge''
  - eat is often followed by an NP
  - British food is not too popular
- N-gram models can be trained by counting and normalization

Example from Daniel Martin book

I want chines

food.

	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.

unigram probabilities):

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

Here are a few other useful probabilities:

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

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

i want english food </s>)

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

Laplace smoothing(add one s

- Add one to all n-gram counts before they are normalized into probabilities
- Not the highest-performing technique for language modeling, but a useful baseline
  - Practical method for other text classification tasks
- $P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}$

Corpus Statistics:

Bigram	Frequency
Chicago Chicago	0+1
Chicago is	2+1
Chicago cold	0+1
Chicago hot	0+1

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

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

Bigram	Frequency
Chicago is	2+1
is cold	4+1
is hot	0+1
…	0+1

Unigram	Probability
Chicago	$\frac{5}{22} = 0.23$
is	$\frac{9}{22} = 0.41$
cold	$\frac{7}{22} = 0.32$
hot	$\frac{1}{22} = 0.05$

Bigram	Probability
Chicago is	$\frac{3}{4+4} = \frac{3}{8} = 0.38$
is cold	$\frac{5}{8+4} = \frac{5}{12} = 0.42$
is hot	$\frac{1}{8+4} = \frac{1}{12} = 0.08$

Probabilities: Before and After smoothing

Bigram	Probability
Chicago is	$\frac{2}{4} = 0.50$
is cold	$\frac{4}{8} = 0.50$
is hot	$\frac{0}{8} = 0.00$

Bigram	Probability
Chicago is	$\frac{3}{8} = 0.38$
is cold	$\frac{5}{12} = 0.42$
is hot	$\frac{1}{12} = 0.08$

Laplace smoothing(add one smoothing)

To keep a language model from assigning zero probability to these unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen.
This modification is called **smoothing or discounting**

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

$$P^{*}_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \qquad (3.23)$$

Thus, each of the unigram counts given in the previous section will need to be augmented by $V = 1446$. The result is the smoothed bigram probabilities in Fig. 3.6.

	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 3.6 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.

Other smoothing techniques

- Backoff: Use the specified n-gram size to estimate probability if its count is greater than 0; otherwise, backoff to a lower-order n-gram.

- Interpolation: Mix the probability estimates from multiple n-gram sizes, weighing and combining the n-gram counts

Backoff

- If the n-gram we need has zero counts, approximate it by backing off to the (n-1)-gram
- Continue backing off until we reach a size that has non-zero counts
- Just like with smoothing, some probability mass from higher-order n-grams need to be

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

- Incorporate a function to distribute probability mass to lower-order ngrams

Kneser-Ney Smoothing

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

Normalizing constant to distribute the probability mass that's been discounted

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

Normalized discount

Number of word types that can follow w_{i-1}

Problem with add-one smoothing

- Data from the AP from (Church and Gale, 1991)
  - Corpus of 44,000,000 bigram tokens, 22,000,000 for training
    - Vocabulary of 273,266 words, i.e. 74,674,306,760 possible bigrams
  - 74,671,100,000 bigrams were unseen
  - frequency is the number of occurrences per 22,000,000 samples
    - To get probability, divide frequency by 22,000,000
  - each unseen bigram was given a <u>frequency</u> of 0.000295

num. of times appeared in training corpus

Freq.observed in testing corpus

Add-one smoothed freq. given to testing corpus

f_{MLE}	$f_{empirical}$	$f_{add\text{-}one}$
0	0.000027	0.000295
1	0.448	0.000589
2	1.25	0.000884
3	2.24	0.001180
4	3.23	0.001470
5	4.21	0.001770

too high

too low

Adding a little of probability over a huge number of unseen events gives too much probability mass to all unseen events

Instead of giving small portion of probability to unseen events, most of the probability space is given to unseen events

http://www.csd.uwo.ca/courses/CS4442b/L9-NLP-LangModels.pdf

Add K smoothing

If k=0.5, Lidstone's law is called Expected Likelihood estimation or Jeffrey's Perks law.

- instead of adding 1, add some other (smaller) positive value δ

$$P_{AddD}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + \delta}{N + \delta B}$$

- most widely used value for δ = 0.5
- if δ =0.5, Lidstone's Law is called:
  - the **Expected Likelihood Estimation** (ELE)
  - or the **Jeffreys-Perks** Law

$$P_{ELE}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + 0.5}{N + 0.5B}$$

- better than add-one, but still not very good

Smoothing: Good Turing

- Imagine you are fishing
  - You have bass, carp, cod, tuna, trout, salmon, eel, shark, tilapia, etc. in the sea
- You have caught 10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel
- How likely is it that next species is new?
  - roughly 3/18, since 18 fish total, 3 unique species
- How likely is it that next is tuna? Less than 2/18
  - 2 out of 18 are tuna, but we have to give some “room” to the new species that we may catch in the future
- Say that there are 20 species of fish that we have not seen yet (bass, shark, tilapia,….)
- The probability of any individual unseen species is $\frac{3}{18 \cdot 20}$
- P(shark)=P(tilapia)= $\frac{3}{18 \cdot 20}$

Smoothing: Good Turing

- How many species (n-grams) were seen once?
  - Let N_1 be the number species (n-grams) seen once
- Use it to estimate for probability of unseen species
  - Probability of new species (new n-gram) is N_1/N
- Let N_0 be the number of unseen species (unseen n-grams). Spreading around the mass equally for unseen n-grams, the probability of seeing any individual unseen species (unseen n-gram) is

$$\frac{N_1}{N \cdot N_0}$$

Smoothing: Good Turing

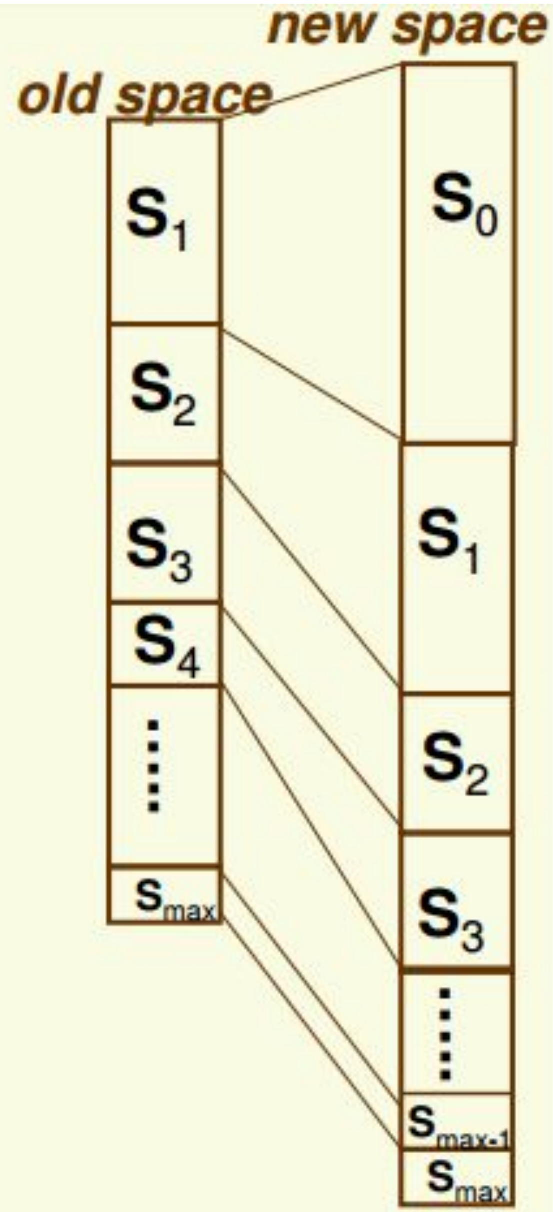
- Back to fishing: you have caught 10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel; 20 species unseen
- How likely is it that next species is new? 3/18
  - The probability of any individual unseen fish is $\frac{3}{18 \cdot 20}$
- What is the new probability of catching a trout?
  - Should be lower than $1/18^{th}$ to make room for unseen fish
  - Idea:
    - if we catch another trout, trout will occur with the rate of 2
    - According to our data, that is the probability of fish with rate 2 (occurring 2 times). Tuna occurs 2 times, so probability is 2/18
    - Now spread the probability of 2/18 over all species which occurred only once – 3 species
    - The probability of catching a fish which occurred 1 time already is $\frac{2}{18 \cdot 3}$

Smoothing: Good Turing

- In general, let **r** be the rate with which an n-gram occurs in the training data
  - Rate is the same thing as count
  - Example: if training data is {"a cow", "a train", "a cow", "do as", "to go", "let us","to go"}, then the rate of "a cow" is 2 and the rate of "let us" is 1
- If an n-gram occurs with rate **r**, we used to get its probability as
  - **r/N**, where **N** is the size of the training data
  - We need to lower all the rates to make room for unseen n-grams
- In general, the number of n-grams which occur with rate **r**+1 is smaller than the number of grams which occur with rate **r**
- Idea: take the portion of probability space occupied by n-grams which occur with rate **r**+1 and divide it among the n-grams which occur with rate **r**

Smoothing: Good Turing

- Let S_r be the n-grams that occur r times in the training data
- Proportion of probability space occupied by n-grams in S_r in the new space = proportion of probability space occupied by n-grams in S_{r+1} in the new space
  - Spread evenly among all n-grams in S_r
- Note no space left for n-grams in S_{max} , has to be fixed



Smoothing: Formula for Good Turing

- N_r be the number different n-grams that we saw in the training data exactly ***r*** times
  - Example: if training data is {"a cow", "a train", "a cow", "do as", "to go", "let us","to go"}, then $N_1 = 3$ and $N_2 = 2$
  - In notation on previous slide, rN_r is the size of S_r
- Probability for any n-gram with rate r is estimated from the space occupied by n-grams with rate r+1
- Let N be the size of the training data.The probability space occupied by n-grams with rate r+1 is:

$$\frac{(r+1)N_{r+1}}{N}$$

- Spread this mass evenly among n-grams with rate r, there are N_r of them

$$\frac{(r+1)N_{r+1}}{N \cdot N_r}$$

- That is for a n-gram x that occurs r times, Good Turing estimate of probability is

$$P_{GT}(x) = (r+1)\frac{N_{r+1}}{N \cdot N_r}$$

Smoothing: Good Turing

$$P_{GT}(w_1 \ldots w_n) = \frac{1}{N} \cdot \underbrace{\frac{(r+1)N_{r+1}}{N_r}}_{r^*}$$

- Another way of looking at Good-Turing:

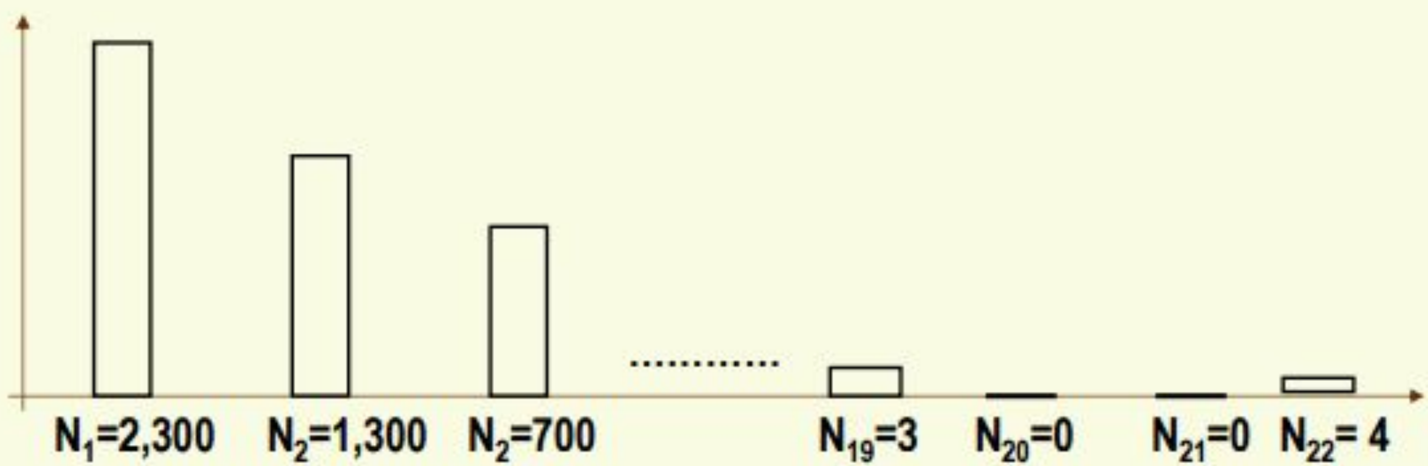
$$P_{MLE}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)}{N} = \frac{r}{N}$$

  - $P_{MLE}(w_1 \ldots w_n) = 0$ for rate $r = 0$, need to increase it
    - at the expense of decreasing the rate of observed nGrams
  - if $r = 0$, new r^* should be larger
  - if $r \neq 0$, new r^* should be smaller

- This is exactly what Good-Turing does
  - For $r = 0$, $r^* = \frac{N_1}{N_0} > r$
  - For $r > 0$, $r^* = \frac{(r+1)N_{r+1}}{N_r}$
  - most likely $r^* < r$ since usually N_{r+1} is significantly less than N_r

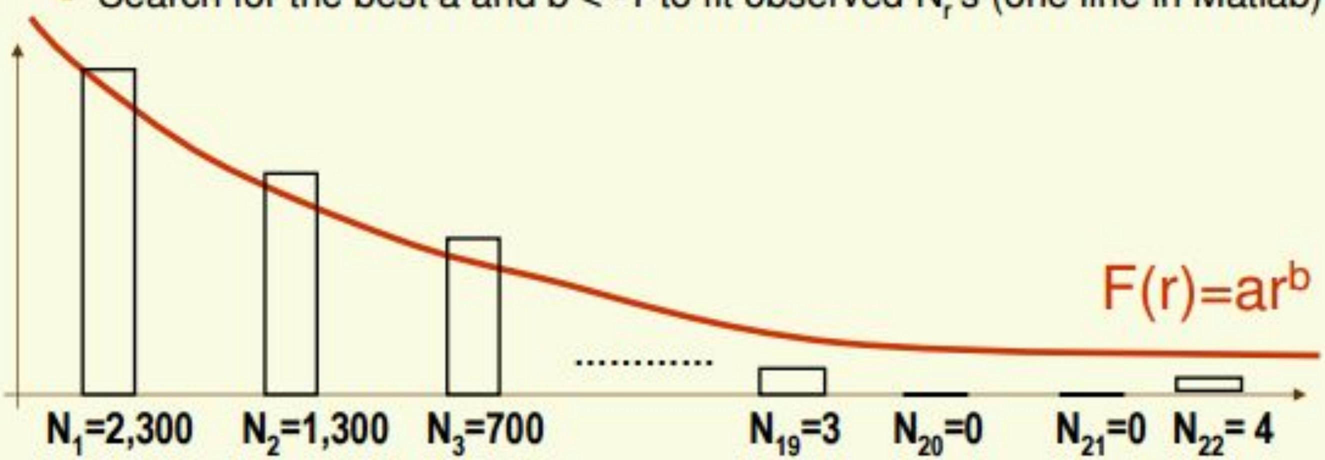
Smoothing: Fixing Good Turing

- That is for an n-gram x that occurs r times, Good Turing estimate of probability is $$P_{GT}(x) = (r+1)\frac{N_{r+1}}{N \cdot N_r}$$
- This works well except for high values of r
  - For high values of r, N_r is not reliable estimate of the number of n-grams that occur with rate r
  - In particular, for the most frequent r it completely fails since $N_{r+1}=0$
- The problem is that N_r is unreliable for high values of r



Smoothing: Fixing Good Turing

- The problem is that N_r is unreliable for high values of r
- Solution 1:
  - use P_{GT} for low values of r, say for r < 10
  - For n-grams with higher rates, use P_{MLE} which is reliable for higher values of r, that is $P_{MLE}(w_1 \ldots w_n)=C(w_1 \ldots w_n)/N$
- Solution 2:
  - Smooth out N_r's by fitting a power law function $F(r)=ar^b$ (with b < -1) and use it when N_r becomes unreliable.
    - Search for the best a and b < -1 to fit observed N_r's (one line in Matlab)



Good Turing vs. Add-One

$r = f_{MLE}$	$f_{empirical}$	f_{Lap}	f_{GT}
0	0.000027	0.000137	0.000027
1	0.448	0.000274	0.446
2	1.25	0.000411	1.26
3	2.24	0.000548	2.24
4	3.23	0.000685	3.24
5	4.21	0.000822	4.22
6	5.23	0.000959	5.19
7	6.21	0.00109	6.21
8	7.21	0.00123	7.24
9	8.26	0.00137	8.25

Smoothing: Fixing Good Turing

- Probabilities will not add up to 1, whether using Solution 1 or Solution 2 from the previous slide
- Have to renormalize all probabilities so that they add up to 1
  - Could renormalize all n-grams
  - Usually we renormalize only the n-grams with observed rates higher than 0
  - Suppose the total space for unseen n-grams is 1/20
  - renormalize the weight of the seen n-grams so that the total is 19/20

Question NPTEL

8) Suppose you are reading an article on Natural Language Processing. Till now, you have read the words "language" - 8 times, "aspect" - 3 times, "processing" - 2 times, "extraction" - 2 times, "question" - once and "dialogue" - once. What are the Maximum Likelihood Estimate (MLE) probability ($P_{processing}$) and Good Turing probability ($P^*_{GT(processing)}$) for reading "processing" as the next word?

1. 1/17, 2/17
2. 2/17, 1/17
3. 3/17, 2/17
4. 2/17, 1.5/17

- 1.
- 2.
- 3.
- 4.

No, the answer is incorrect.
Score: 0

Accepted Answers:
4.

9) With the same setting as Question 8, calculate the MLE and Good Turing probabilities for reading "answering" as the next word:

1. 1/17, 2/17
2. 0, 1/17
3. 0, 2/17
4. 1/17, 1/17

- 1.
- 2.
- 3.
- 4.

No, the answer is incorrect.
Score: 0

Accepted Answers:
3.

Evaluation

Intrisically

Evaluation: How good is our model?

Does our language model prefer good sentences to bad ones?

- Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences?

We train parameters of our model on a

Training on the test set

We can’t allow test sentences into the training set

We will assign it an artificially high probability when we set it in the test set

“Training on the test set”

Bad science!

And violates the honor code

Extrinsic evaluation of N-gram models

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves.

Best evaluation for comparing models A and B

- Put each model in a task
  - spelling corrector, speech recognizer, MT system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

Extrinsic evaluation

- Time-consuming; can take days or weeks

So

- Sometimes use **intrinsic** evaluation: **perplexity**
- Bad approximation
  - unless the test data looks **just** like the training data
  - So **generally only useful in pilot experiments**
- But is helpful to think about.

perplexity example

- Perplexity is a measure of how well a language model predicts a sample. A lower perplexity indicates a better model.

Example 1: Predicting the Next Word

- Given a sentence fragment, the model predicts the next word.
- Sentence Fragment: "The cat sat on the"

Model Predictions:

- Model A: ["mat" (0.6), "sofa" (0.2), "table" (0.1), "floor" (0.1)]
- Model B: ["sofa" (0.4), "mat" (0.3), "table" (0.2), "floor" (0.1)]

True Next Word: "mat"

- **Model A Perplexity Calculation**:
  - Probability of "mat" = 0.6
  - Perplexity = $\frac{1}{P(\text{"mat"})} = \frac{1}{0.6} \approx 1.67$
- **Model B Perplexity Calculation**:
  - Probability of "mat" = 0.3
  - Perplexity = $\frac{1}{P(\text{"mat"})} = \frac{1}{0.3} \approx 3.33$

Model A has a lower perplexity, indicating it predicts the next word better.

Transformer code

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
# Load pre-trained model and tokenizer
model_name = 'gpt2'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
# Set the model to evaluation mode
model.eval()
# Text to evaluate
text = "The quick brown fox jumps over the lazy dog."
# Tokenize the input text
inputs = tokenizer(text, return_tensors='pt')
input_ids = inputs['input_ids']
# Ensure the model doesn't update its weights
with torch.no_grad():
    outputs = model(input_ids, labels=input_ids)
    loss = outputs.loss
# Calculate perplexity
perplexity = torch.exp(loss).item()
print(f"Perplexity: {perplexity}")
```

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

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

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

Chain rule:

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

For bigrams:

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

Minimizing perplexity is the same as maximizing probability

Perplexity as branching factor

Let's suppose a sentence

consisting of random digits

What is the perplexity of this

sentence according to a model

that assign P=1/10 to each digit?

$$\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}$$

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

- Bag of words: Extracts features from the text
- TF-IDF: Information retrieval, keyword extraction
- Word2Vec: Semantic analysis task
- GloVe: Word analogy, named entity recognition tasks
- BERT: language translation, question answering system

https://www.turing.com/kb/guide-on-word-embeddings-in-nlp