# CS4442/9542b
# Artificial Intelligence II
# prof. Olga Veksler

*Lecture 14*

*Natural Language Processing*

Language Models

Many slides from: Joshua Goodman, L. Kosseim, D. Klein, D. Jurafsky

# Outline

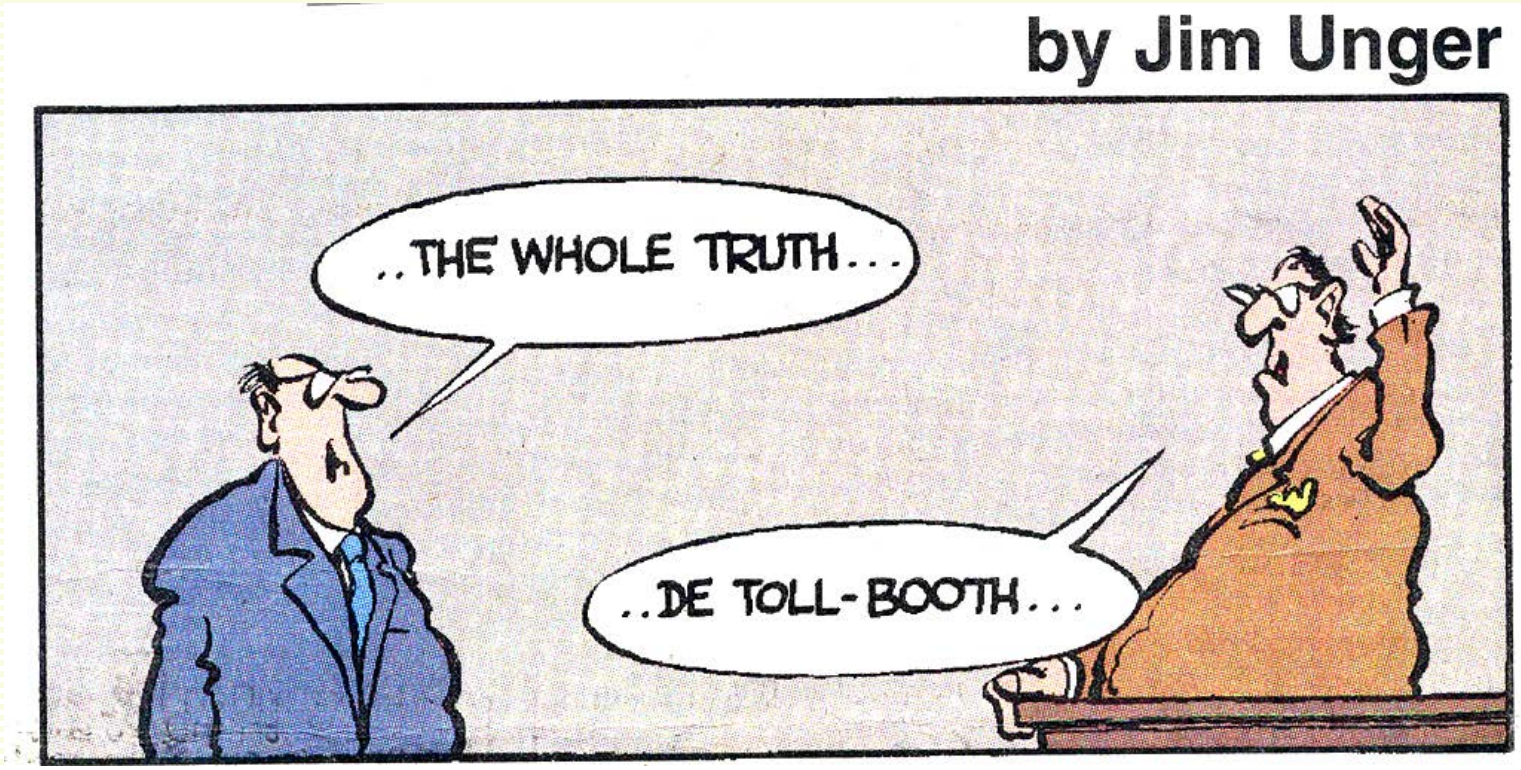
- Why we need to model language
- Probability background
  - Basic probability axioms
  - Conditional probability
  - Chain Rule
- n-gram model
- Parameter Estimation Techniques
  - MLE
  - Smoothing
    - Add-One, add-Delta
    - Good-Turing

# Why Model Language?

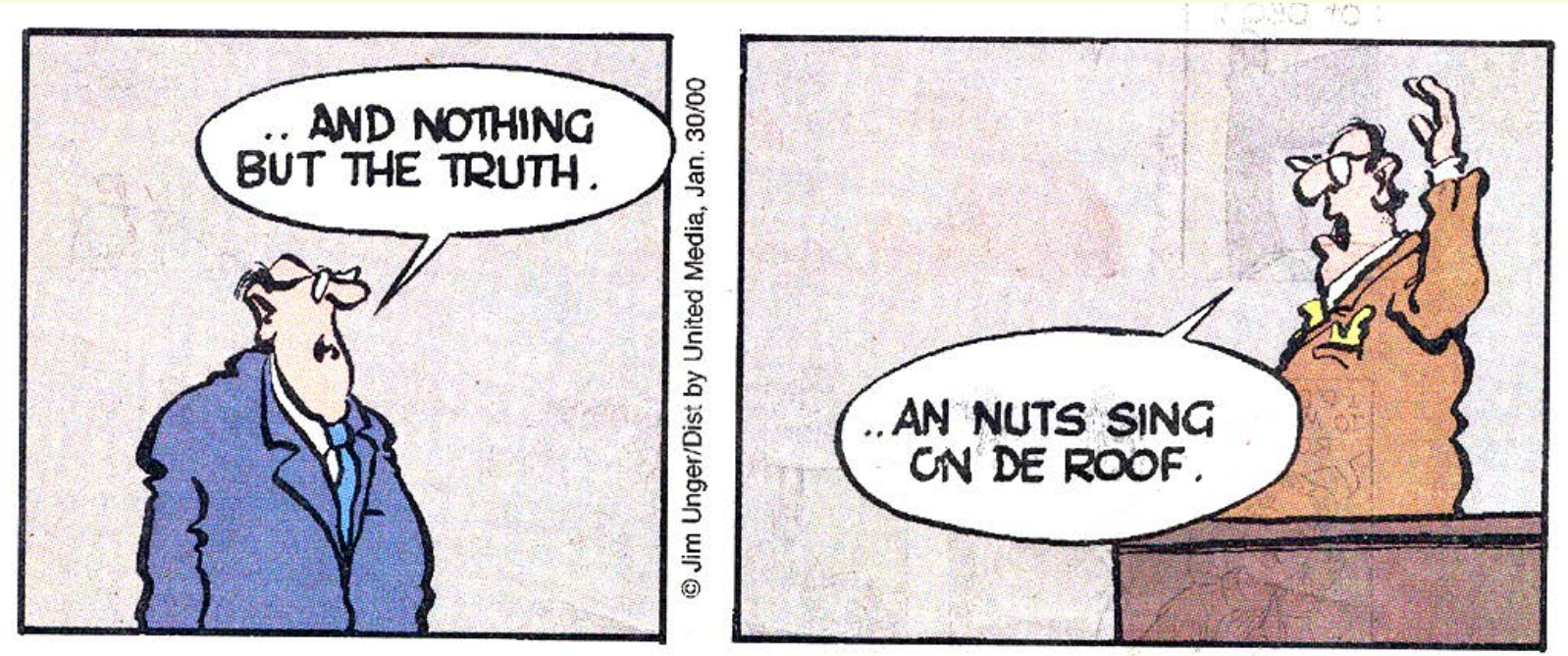
- Design probability model **P**() such that
- Spell checker:
  - **P**(I think there are OK) < **P**(I think they are OK)
- Speech recognition:
  - **P**(lie cured mother) < **P**(like your mother)
- Optical character recognition
  - **P**(thl cat) < **P**(the cat)
- Machine translation: "On voit Jon à la télévision"
  - **P**(In Jon appeared TV) < **P**(Jon appeared on TV)
- Many other applications

# Language Model for Speech Recognition

# Language Model for Speech Recognition



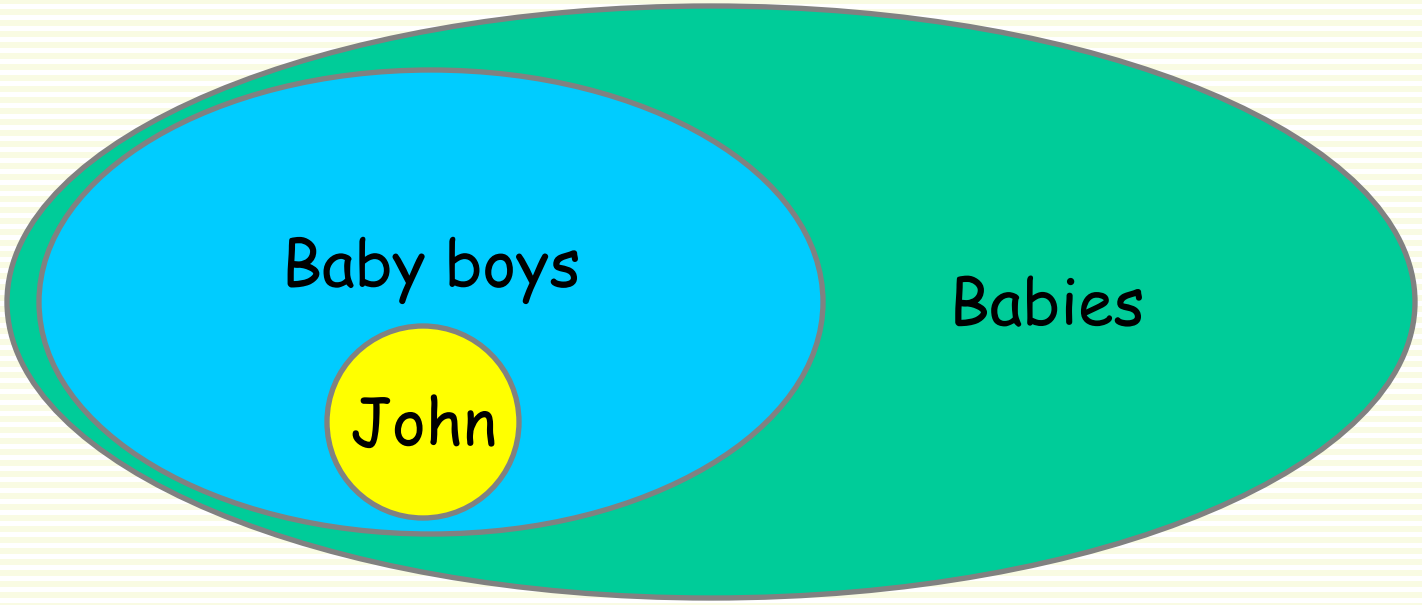
# Language Model for Speech Recognition



# Language Model for Speech Recognition

# Basic Probability

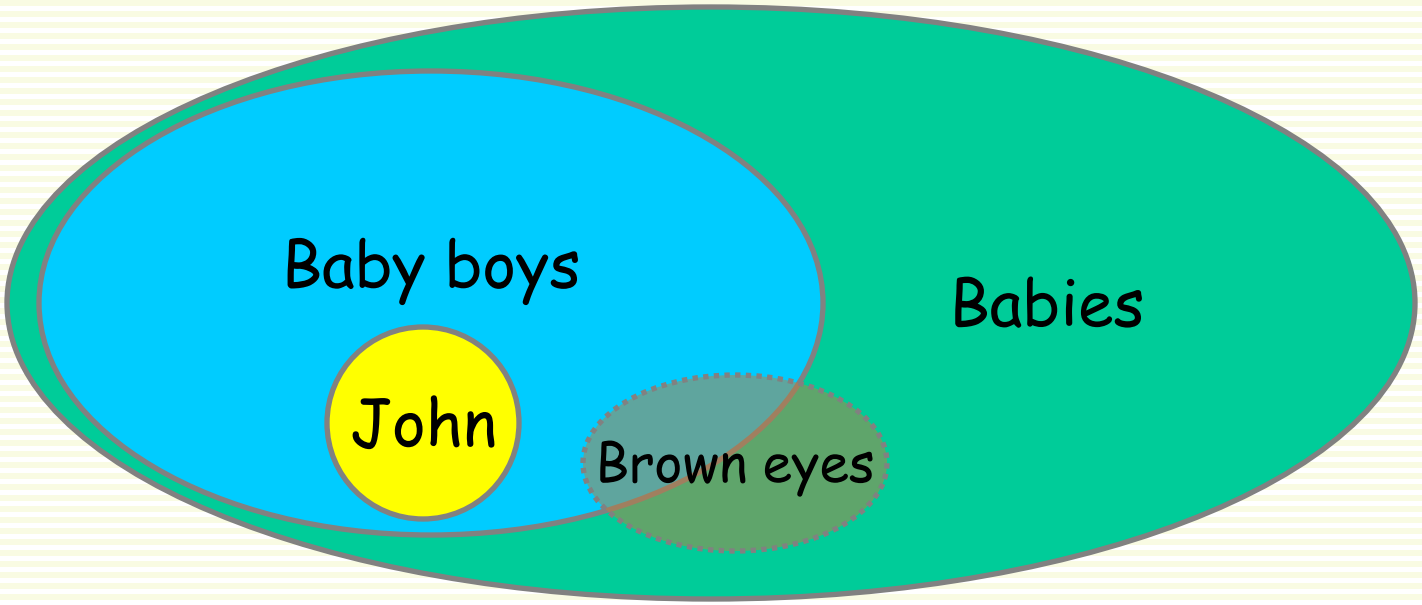
- **P**(X) is probability that X is true
  - **P**(baby is a boy) = 0.5 (1/2 of all babies are boys)
  - **P**(baby is named John) = 0.001 (1 in1000 babies  named John)



# Joint probabilities

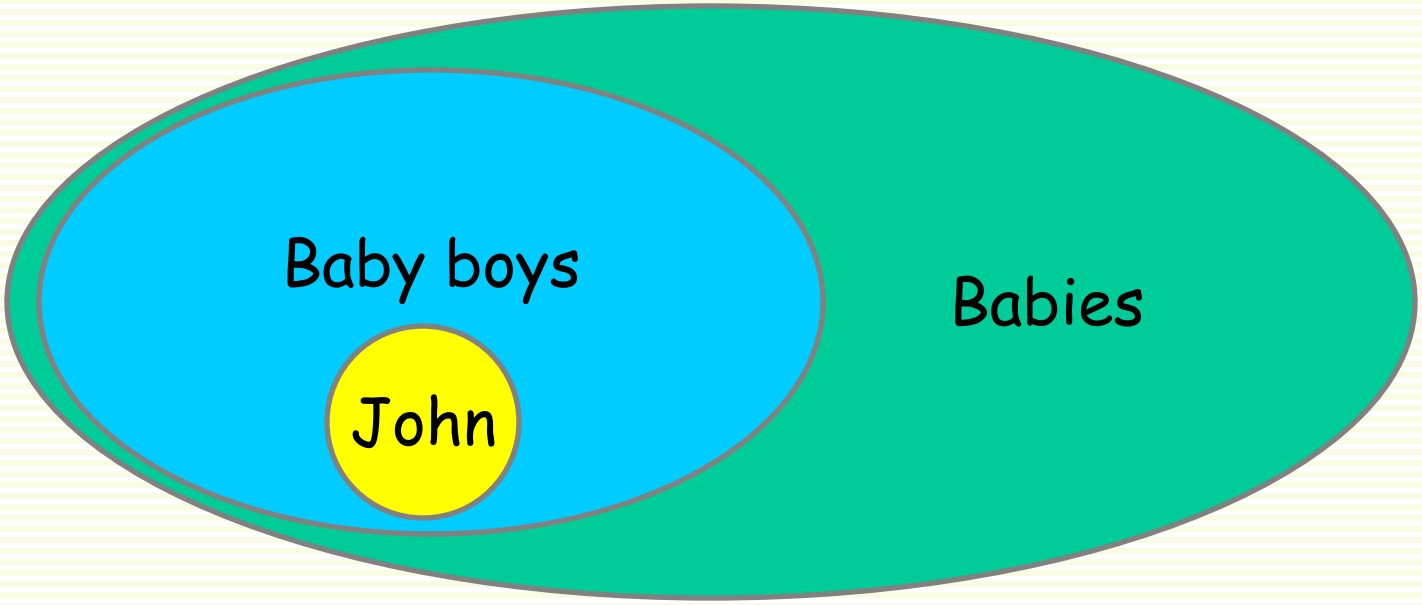
- **P**(X,Y) is probability that X and Y are both true

  **P**(brown eyes, boy) = (number of all baby boys with brown eyes)/(total number of babies)



# Conditional probability

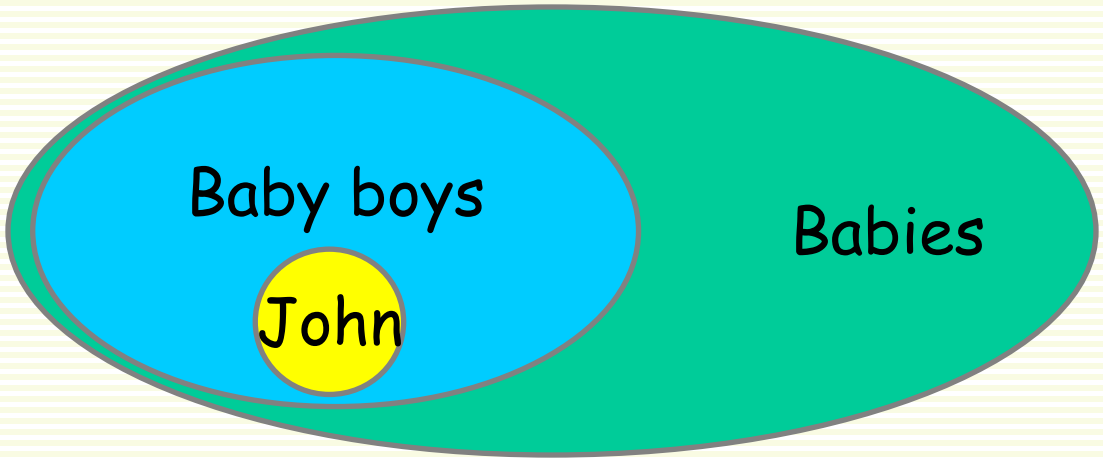
- **P**(X|Y) is probability that X is true when we already know Y is true



# Conditional Probability

- **P**(X|Y) = **P**(X, Y) / **P**(Y)
- **P**(baby is named John | baby is a boy) =

$$\frac{\textbf{P}(\text{baby is named John, baby is a boy})}{\textbf{P}(\text{baby is a boy})} = \frac{0.001}{0.5} = 0.002$$




- **P**(baby is a boy | baby is named John ) = 1

# Chain Rule

- Conditional Probability

$$\mathbf{P}(Y|X) = \mathbf{P}(X,Y) / \mathbf{P}(X)$$

- Rewrite

$$\mathbf{P}(X,Y) = \mathbf{P}(Y|X)\,\mathbf{P}(X)$$

- Extend to three events

$$\mathbf{P}(X,Y,Z) = \mathbf{P}(Y,Z|X)\mathbf{P}(X) = \mathbf{P}(Z|X,Y)\mathbf{P}(Y|X)\mathbf{P}(X)$$

- Extend to multiple events

$$\mathbf{P}(X_1,X_2,\ldots,X_n) = \mathbf{P}(X_1)\mathbf{P}(X_2|X_1)\mathbf{P}(X_3|X_1X_2)\ldots\mathbf{P}(X_n|X_1,\ldots,X_{n-1})$$

# Language Modeling

- Start with vocabulary
  - words vocabulary V = {a, an, apple,…, zombie}
  - or character vocabulary V = {a, A,…., z, Z,*,…, -}
- In LM, events are sequences of words (or characters)
- Example “an apple fell” or “abracadabra!!!+”
- **P**(an apple fell) is the probability of the joint event that
  - the first word in a sequence is “an”
  - the second word in a sequence is “apple”
  - the third word in a sequence is “fell”
- **P**( fell | an apple ) is probability that the third word in a sequence is “fell” given that the previous 2 words are “an apple”

# Probabilistic Language Modeling

- A language model is a probability distribution over word or character sequences

$$\mathbf{P}(W) = P(w_1 w_2 w_3 w_4 w_5 \ldots w_k)$$

- Want:
  - **P**("And nothing but the truth") $\approx 0.001$
  - **P**("And nuts sing on the roof") $\approx 0.000000001$
- Related task: probability of an upcoming word:

$$\mathbf{P}(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these:

$$\mathbf{P}(W) \quad \text{or} \quad \mathbf{P}(w_k | w_1, w_2 \ldots w_{k-1})$$

  is called a **language model**
- Build model **P** from observed texts (corpora)

# Probabilistic Language Modeling

- Get lots of training text (corpora)
- Use it to estimate $P(w_1 w_2 w_3 w_4 w_5 \ldots w_k)$ :
- Naïve idea: $$\mathbf{P}(\mathbf{w_1 w_2 \ldots w_k}) = \frac{\mathbf{C}(\mathbf{w_1 w_2 \ldots w_k})}{\mathbf{N}}$$
  - where $C(w_1 w_2 \ldots w_k)$ is the number of times (count) sentence $w_1 w_2 \ldots w_k$ appears in training data
  - N is number of sentences in training data
- Problem: language is infinite, many reasonable English sentences do not appear in training data
  - "A happy new mother first put on a purple polka-dot dress on her baby daughter, and then kissed her tiny left toe. "
  - Do not want any sentence to have probability 0

# Markov Assumption

- Can we make some simplifying assumptions?
- Consider

  **P**(computer | instead of listening to this boring lecture, I would like to play on my )

- Probability that "computer" follows "Instead of listening to this boring lecture, I would like to play on my" is intuitively almost the same as probability that "computer" follows words "play on my"
- Probability of the next word depends most strongly on just a few previous words

# Shannon Game (1951)

| Context | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Entropy (H) | 4.76 | 4.03 | 3.21 | 3.1 |

*"I am going to make a collect …"*

- Predict next word/character given *n-1* previous words/characters
- Human subjects were shown a few characters of text and were asked to guess the next character
- As context increases, entropy decreases
  - the smaller the entropy => the larger probability of predicting next letter
- But only a few words is enough to make a good prediction on the next word, in most cases
- Evidence that we only need to look back at n-1 previous words

# n-grams

- n-gram model: probability of a word depends only on the n-1 previous words (the history)

$$\mathbf{P}(w_k \mid \cancel{w_1 w_2 \ldots w_{k-n}} w_{k+1-n} \ldots w_{k-1}) \approx \mathbf{P}(w_k \mid w_{k+1-n} \ldots w_{k-1})$$

- This called **Markov Assumption**: only the closest n words are relevant
- Special cases:
  - Unigram (n=1): previous words do not matter
  - Bigram (n=2): only the previous one word matters
  - Trigram (n=3): only the previous two words matter

# Example: Trigram Approximation (n = 3)

- Each word depends only on previous two words
  - three words total with the current one
  - tri means three
  - gram means writing

- **P**(the|… whole truth and nothing but) ≈
  **P**(the|nothing but)
- **P**(truth|… whole truth and nothing but the) ≈
  **P**(truth|but the)

# Chain Rule

- First Decompose using the chain rule

  **P**(and nothing but the truth) =

  **P**(and)

  x **P**(nothing|and)

  x **P**(but|and nothing)

  x **P**(the|~~and~~ nothing but)

  x **P**(truth|~~and nothing~~ but the)

- **P**(and nothing but the truth) ≈

  **P**(and)**P**(nothing|and) **P**(but|and nothing)
  **P**(the|nothing but) **P**(truth|but the)

# How Compute Trigram Probabilities?

- $\mathbf{P}(w_3 \mid w_1\, w_2) \approx ?$
  - these probabilities are usually called *parameters*

- First rewrite

$$\mathbf{P}(w_3 \mid w_1\, w_2) = \frac{\mathbf{P}(w_1\, w_2\, w_3)}{\mathbf{P}(w_1\, w_2)}$$

- Need to estimate $\mathbf{P}(w_1\, w_2\, w_3)$, $\mathbf{P}(w_1\, w_2)$, $\mathbf{P}(w_1)$, etc.
  - call these trigram, bigram, unigram, etc.
- Get lots of real text, and approximate based on counts

$$\mathbf{P}(w_1\, w_2 w_3) = \frac{\mathbf{C}(\mathbf{w_1\, w_2\, w_3})}{\text{number of trigrams in text}}$$

  - $\mathbf{C}(w_1\, w_2\, w_3)$ is number of times $\mathbf{w_1\, w_2\, w_3}$ observed in training text

# Small Trigram Example

- Training text

  "and nothing but the truth when nuts and nothing on the roof"

- Training text has 12 unigrams, 11 bigrams, 10 trigrams

$$\mathbf{P}(\text{but} \mid \text{and nothing}) = \frac{\mathbf{P}(\text{and nothing but})}{\mathbf{P}(\text{and nothing})} = \frac{1/10}{2/11} = \frac{11}{20}$$

$$\mathbf{P}(\text{and nothing but}) = \frac{\mathbf{C}(\text{and nothing but})}{10} = \frac{1}{10}$$

$$\mathbf{P}(\text{and nothing}) = \frac{\mathbf{C}(\text{and nothing})}{11} = \frac{2}{11}$$

# How Compute Trigram Probabilities?

- In practice in a file with **N** words
  - **N** unigrams
  - **N**-1 bigrams
  - **N**-2 trigrams, etc.
- For large **N** dividing by **N**, or **N**-1, or **N**-2 makes no difference in practice

$$5/10{,}000{,}006 =_{\text{almost}} 5/10{,}000{,}005 =_{\text{almost}} 5/10{,}000{,}004$$

- For previous example

$$\mathbf{P}(\text{but} \mid \text{and nothing}) = \frac{1/12}{2/12} = \frac{1}{2}$$

# How Compute Trigram Probabilities?

- Calculations simplify

$$\mathbf{P}(w_3 \mid w_1 w_2) = \frac{\mathbf{C}(w_1 w_2 w_3)/\mathbf{N}}{\mathbf{C}(w_1 w_2)/\mathbf{N}} = \frac{\mathbf{C}(w_1 w_2 w_3)}{\mathbf{C}(w_1 w_2)}$$

- This also avoids **P** > 1
- Consider training text again

"and nothing but the truth when nuts and nothing on the roof"

- With exact arithmetic, i.e. **N**-2, **N**-1

$$\mathbf{P}(\text{truth} \mid \text{but the}) = \frac{\mathbf{C}(\text{but the truth})\ /10}{\mathbf{C}(\text{but the})\ /11} = \frac{1/10}{1/11} = \frac{11}{10}$$

# Computing Trigrams

- From now on

$$\mathbf{P}(w_3 \mid w_1 w_2) = \frac{\mathbf{C}(w_1 w_2 w_3)}{\mathbf{C}(w_1 w_2)}$$

$$\mathbf{P}(w_1 w_2 w_3) = \frac{\mathbf{C}(w_1 w_2 w_3)}{\mathbf{N}}$$

- where **N** is number of words in training text

# Trigrams, continued

- **P**(and nothing but the truth) ≈

  **P**(and)**P**(nothing|and) **P**(but|and nothing)

  **P**(the|nothing but) **P**(truth|but the)

$$= \frac{\mathbf{C}(\text{and})}{\mathbf{N}} \frac{\mathbf{C}(\text{and nothing})}{\mathbf{C}(\text{and})} \frac{\mathbf{C}(\text{and nothing but})}{\mathbf{C}(\text{and nothing})} \frac{\mathbf{C}(\text{nothing but the})}{\mathbf{C}(\text{nothing but})} \frac{\mathbf{C}(\text{but the truth})}{\mathbf{C}(\text{but the})}$$

- **N** is the number of words in training text

# Text Generation with n-grams

- Trained on 40 million words from WSJ (wall street journal)
- Generate next word according to the n-gram model
- Unigram
  - *Months the my and issue of year foreign new exchange's September were recession exchange new endorsed a acquire to six executives.*
- Bigram
  - *Last December through the way to preserve the Hudson corporation N.B.E.C. Taylor would seem to complete the major central planner 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.*
- Trigram
  - *They also point to ninety 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.*

# Example with Sentence Start/End

- Training text:

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

- Bigram model: $\mathbf{P}(w_i|w_{i-1}) = \frac{\mathbf{C}(w_{i-1}w_i)}{\mathbf{C}(w_{i-1})}$

- Some bigram probabilities

$\mathbf{P}(I|<s>) = 2/3 = 0.67$ $\quad$ $\mathbf{P}(Sam|<s>) = 1/3 = 0.33$

$\mathbf{P}(</s>|Sam) = 1/2 = 0.5$ $\quad$ $\mathbf{P}(Sam|am) = 1/2 = 0.5$

$\mathbf{P}(am|I) = 2/3 = 0.67$ $\quad$ $\mathbf{P}(do|I) = 1/3 = 0.33$

# Raw Bigram Counts

- Can construct **V**-by-**V** matrix of probabilities/frequencies
- **V** = size of the vocabulary we are modeling
- Used 922 sentences

2nd word

| 1st word | 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 |

# Bigram Probabilities

- Normalize by unigrams to get conditional P(second|first)

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

- Result

P(want|I)

| | 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 |

P(<s> I want chinese food </s>) = P(I|<s>)
× P(want|I)
× P(chinese|want)
× P(food|chinese)
× P(</s>|food)
= .000031

# Practical Issue

- Use log space
  - to avoid underflow
  - also adding is faster than multiplying
  - instead of **P**(a)×**P**(b)×**P**(c) compute log[**P**(a)]+ log[**P**(a)] +log[**P**(a)]
- Example, instead of

  **P**(<s> I want chinese food </s>) =**P**(I|<s>) × **P**(want|I) × **P**(chinese|want)
  × **P**(food|chinese) × **P**(</s>|food)
  = .000031
- Compute

  log[**P**(<s> I want chinese food </s>)] = log[**P**(I|<s>)] + log[**P**(want|I)] +
  log[**P**(chinese|want)] + **P**(food|chinese)
  + log[**P**(</s>|food)] = -4.501

# Google N-Gram Release, August 2006

AUG
3

## All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of *one trillion words* from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Watch for an announcement at the Linguistics Data Consortium (LDC), who will be distributing it soon, and then order your set of 6 DVDs. And let us hear from you - we're excited to hear what you will do with the data, and we're always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

# Which n-gram to Use?

- *"the large green _____ ."*
  - *"mountain"*? *"tree"*? *"pill"*? *"broccoli"*? …
- *"Sue swallowed the large green _____ ."*
  - *"pill"*? *"broccoli"*?
- Knowing that Sue "*swallowed*" helps narrow down possibilities
- Larger ***n*** gives more context, i.e. looking back further
- But need more data to estimate larger **n-**grams reliably

# Which n-gram to use?

- Example: for a vocabulary of 20,000 words
  - number of bigrams = 400 million ($20\,000^2$)
  - number of trigrams = 8 trillion ($20\,000^3$)
  - number of four-grams = $1.6 \times 10^{17}$ ($20\,000^4$)
- number of **n**-grams is number of parameters to learn
- Going from **n**-gram to (**n**+1)-gram, number of parameters grows by a factor of **n**
- For reliable estimates, need much more data
- But usually training data has fixed size of **N** words, it does not change when we go from **n**-gram to (**n**+1)-gram

# Unigram vs. Bigram Illustration

- Suppose
  - vocabulary size is 10,000=$10^4$
  - training text has 100,000 = $10^5$ words
- For Unigrams
  - need to estimate unigram counts for each vocabulary word, i.e. C('place'), C('apple'), etc.
  - number of parameters (unigram counts) to estimate is $10^4$
  - on average, $10^5$/ $10^4$ = 10 training samples per parameter, reasonable
- For Bigrams
  - need to estimate bigram counts for $10^4$* $10^4$ = $10^8$ possible bigrams
  - number of parameters (bigram counts) to estimate is $10^8$
  - number of training samples is still $10^5$
  - on average, have $10^5$ /$10^8$ =$10^{-3}$ training samples to fit per parameter
  - highly insufficient, need much more data

# Reliability vs. Discrimination

- Larger n
  - **greater discrimination**: more information about the context
  - but **less reliability**:
    - cannot estimate parameters reliably from limited data (data sparseness)
- Smaller n
  - **less discrimination**, not enough history to predict next word very well model is not as accurate
  - but **more reliability**
    - more instances in training data, better statistical estimates of parameters
- Bigrams or trigrams are most often used in practice
  - works well, although there are longer-range dependencies not captured

# Reducing number of Parameters

- with a 20 000 word vocabulary
  - bigram needs to store 400 million parameters
  - trigram needs to store 8 trillion parameters
  - using a language model > trigram is impractical
- to reduce the number of parameters
  - use stems instead of word types
    - help = helps = helped
  - group words into semantic classes
    - {Monday,Tuesday,Wednesday,Thursday,Friday} = one word
  - seen once --> same as unseen

# Statistical Estimators

- Count-based parameter estimation is called **Maximum Likelihood Estimation (MLE)**
  - Not reliable due to data sparseness
- Smoothing
  - Add-one -- Laplace
  - Add-delta
    - Lidstone's
    - Jeffreys-Perks' Laws (ELE)
  - Good-Turing

# Maximum Likelihood Estimation

- Let $C(w_1...w_n)$ be the frequency of n-gram $w_1...w_n$

$$P_{MLE}(w_n \mid w_1...w_{n-1}) = \frac{C(w_1...w_n)}{C(w_1...w_{n-1})}$$

- Maximizes probability of the **training** corpus
- However, interested in good performance on **test** data

# Data Sparseness Example

- In training corpus, have 10 instances of "*come across*"
  - 8 times, followed by "*as*"
  - 1 time, followed by "*more*"
  - 1 time, followed by "*a*"
- Therefore
  - $P_{MLE}(\text{as} \mid \text{come across}) = \frac{C(\text{come across as})}{C(\text{come across})} = \frac{8}{10}$
  - $P_{MLE}(\text{more} \mid \text{come across}) = 0.1$
  - $P_{MLE}(\text{a} \mid \text{come across}) = 0.1$
  - $P_{MLE}(X \mid \text{come across}) = 0$ where $X \neq$ "*as*", "*more*", "*a*"

# Problem with MLE: Data Sparseness

- From [Balh et al 83]
  - training with 1.5 million words
  - 23% of the trigrams from another part of the same corpus were previously unseen
- in Shakespeare's work
  - out of all possible bigrams, 99.96% were never used
- So MLE alone is not good enough estimator

# Problem with MLE: Data Sparseness

- MLE assigns a probability of zero to unseen events
- Most likely no trigram “and nuts sing” in training data
- Therefore estimate P(and nuts sing) = 0
- Any sentence which has “and nuts sing” has probability 0
  - want P(“and nuts sing”) to be small, but not 0
- Even for unigram, probability of any unigram involving an unseen word will be zero
- but … most words are rare
- n-grams involving rare words are even more rare… data sparseness

# Discounting or Smoothing

- MLE alone is unsuitable for NLP because of the sparseness of the data
- We need to allow for possibility of seeing events not seen in training
- Must use a **Discounting** or **Smoothing** technique
- Decrease the probability of previously seen events to give a little bit of probability for previously unseen events

# Smoothing

- Increase **P**(unseen event) → decrease **P**(seen event)
- **P**(w|denied the)

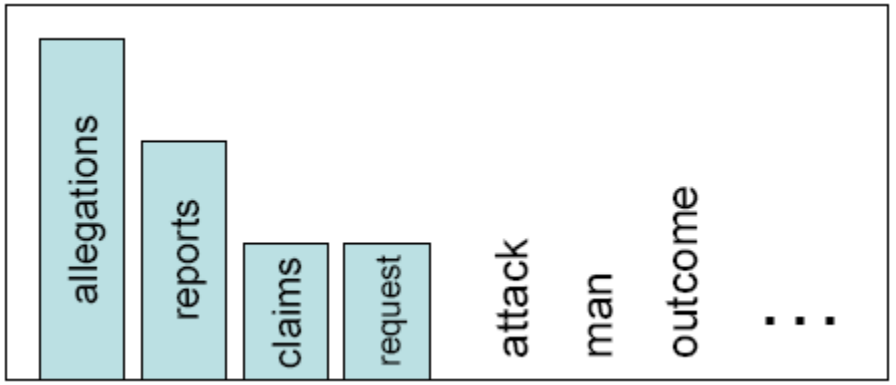
  3 allegations

  2 reports

  1 claims

  1 request

  7 total




- Smoothing flattens spiky distributions so they generalize better

  **P**(w|denied the)

  2.5 allegations

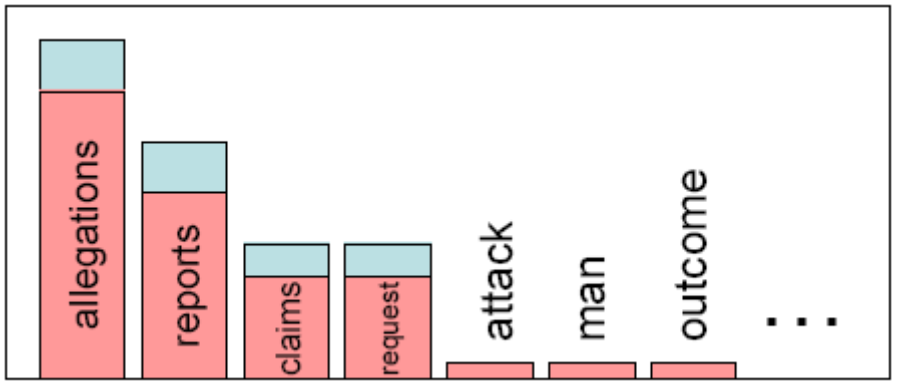
  1.5 reports

  0.5 claims

  0.5 request

  2 other

  7 total



# Many smoothing techniques

- Add-one
- Add-delta
- Good-Turing smoothing
- Many others

# Add-one Smoothing (Laplace's Law 1814)

- Give a little bit of probability space to unseen events
- Pretend we have seen every n-gram at least once
- Intuitively appended all possible n-grams to training data

| real data | fake data |
|---|---|
| **N** bigrams | all possible bigrams |

- Training data has **N** n-grams
- The "new" size is **N+B**,where **B** is # of all possible n-grams
- If **V** words in vocabulary, then:
  - **B**= **V***V** for bigrams
  - **B**=**V***V***V** for trigrams
  - etc.
- We get: $$P_{Add1}(w_1\, w_2 \ldots w_n) = \frac{C(w_1\, w_2 \ldots w_n) + 1}{N + B}$$

# Add-One Example

- Character model
- Training data = “abraabr”
  - **N** = 7
  - bigrams = ab,br,ra,aa,ab,br
- Let **V** = 256
- With bigram approximation (**n** = 2), B = $256^2$

$$P_{Add1}(w_1 w_2 \ldots w_n) = \frac{C(w_1 w_2 \ldots w_n) + 1}{N + B}$$

$$P_{Add1}(ab) = \frac{C(ab) + 1}{7 + 256^2} = \frac{3}{7 + 256^2} \approx 4.6 \times 10^{-5}$$

# How well does Add-One Smoothing Works?

- Works ok if sparsity problem is mild
  - not a lot of missing nGrams
  - problems if  many missing nGrams

# Example Allocation to Unseen Bigrams

- Data from the AP from (Church and Gale, 1991)
- **N** = 22,000,000
- **V** = 273,266
- **B** = $\mathbf{V}^2$ = 74,674,306,756
- 74,671,100,000 unseen bigrams
- Add One probability of unseen bigram:

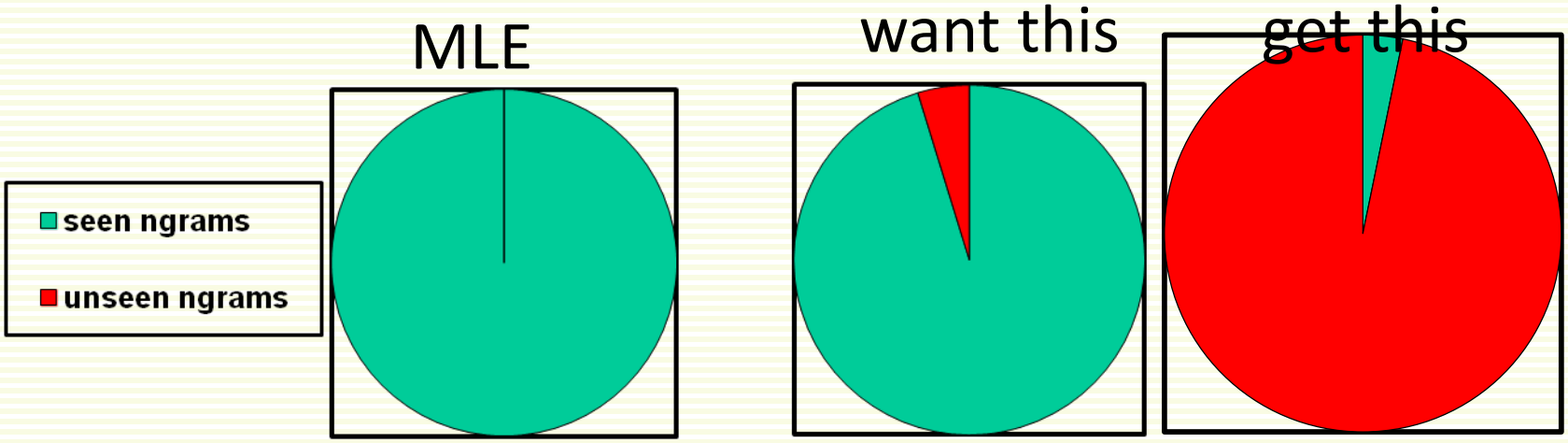
$$\frac{1}{N+B} = \left(\frac{1}{22{,}000{,}000 + 74{,}674{,}306{,}756}\right) = 1.33875\text{x}10^{-11}$$

- Portion of probability mass given to unseen bigrams:

number of unseen bigrams x **P**(unseen bigram) =

$$74{,}671{,}100{,}000 \times \left(1.33875 \times 10^{-11}\right) \approx 99.96$$

# Problem with add-one smoothing




- each individual unseen n-gram is given a low probability
- but there is a huge number of unseen n-grams
- Instead of giving small portion of probability to unseen events, most of the probability space is given to unseen events

# Add-delta smoothing (Lidstone's law)

- instead of adding 1, add some smaller positive value $\delta$

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

- This is called Lidstone's law
- most widely used value for $\delta = 0.5$, in this case it's called
  - the **Expected Likelihood Estimation** (ELE)
  - or the **Jeffreys-Perks** Law

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

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

# Add-Delta Example

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

- Let us use character model
- Training data = "abraabr"
  - **N** = 7
  - trigrams = abr, bra, raa, aab, abr
- Let **V** = 256
- With trigram approximation (**n** = 3), B = $256^3$

$$P_{AddD}(aba) = \frac{C(aba) + 0.1}{7 + 0.1 \cdot 256^3} = \frac{0.1}{7 + 0.1 \cdot 256^3} \approx 5.9 \times 10^{-8}$$

# Evaluation: How good is our model?

- Train parameters of our model on a **training set**
- How to choose delta in Add-Delta smoothing?
- Use validation or cross-validation
- Choose model that maximizes probability of validation data

# Good Turing Smoothing, Preparation

- $\mathbf{N}_r$ is the number of different n-grams in training data occurring exactly **r** times
- Example
  - training data =“catch a cow, make a cow sing, make a cow dance“
  - bigrams
    - “catch a” occurs 1 time
    - “a cow” occurs 3 times
    - “cow make” occurs 1 time
    - “make a” occurs 2 times
    - “cow sing” occurs 1 time
    - “sing make” occurs 1 time
    - “cow dance” occurs 1 time
  - $N_1 = 5$ , $N_2 = 1$, $N_3 = 1$
  - Assume vocabulary

    V = {a, cow, catch, sing, make, dance, UNKNOWN}
  - Then $\mathbf{N_0} = 7^2 - (N_1 + N_2 + N_3) = 42$

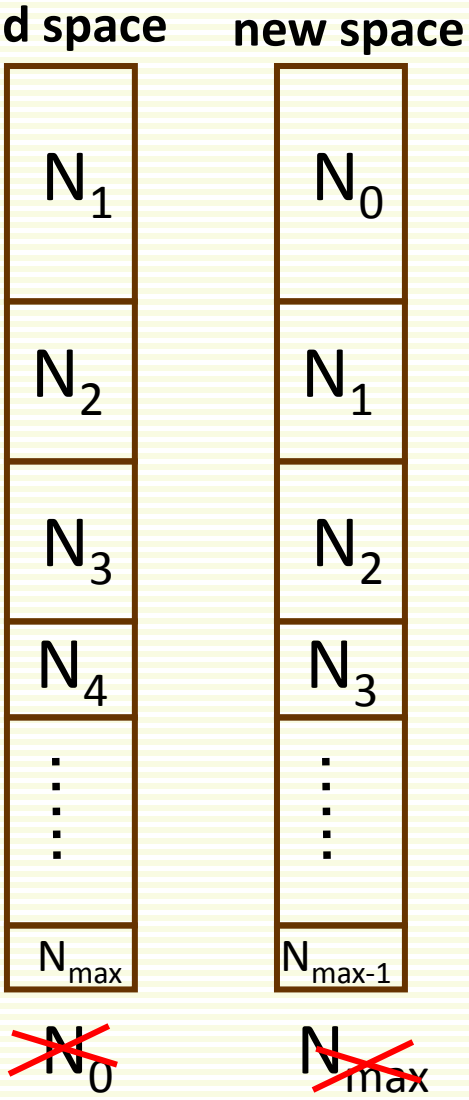
# Good Turing Smoothing, 1953

- $N_r$ number of different n-grams that occur **r** times
- Under MLE, $N_0$ has no probability space
- Good Turing idea
  - $N_0$ steals space from $N_1$
  - $N_1$ steals space from $N_2$
  - $N_2$ steals space from $N_3$
  - etc…
- Thus probability for any **n**-gram with rate **r** is estimated from space occupied by **n**-grams with rate **r**+1
- Space occupied by **n**-grams with rate **r**+1 is

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

- Spread it evenly among **n**-grams with rate **r**

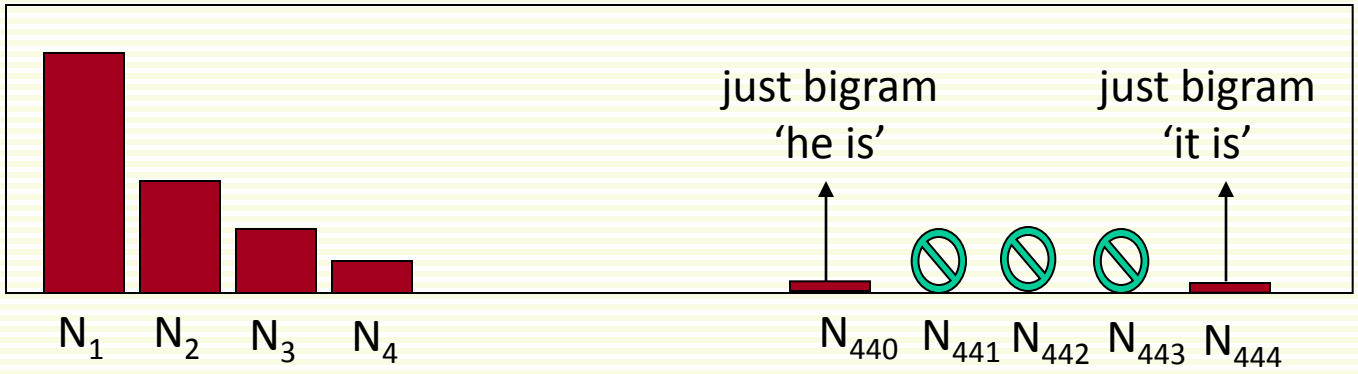
$$\frac{(r+1)N_{r+1}}{N_r N}$$




- If n-gram x has rate **r**, Good Turing estimate is $P_{GT}(x) = (r+1)\frac{N_{r+1}}{N \cdot N_r}$

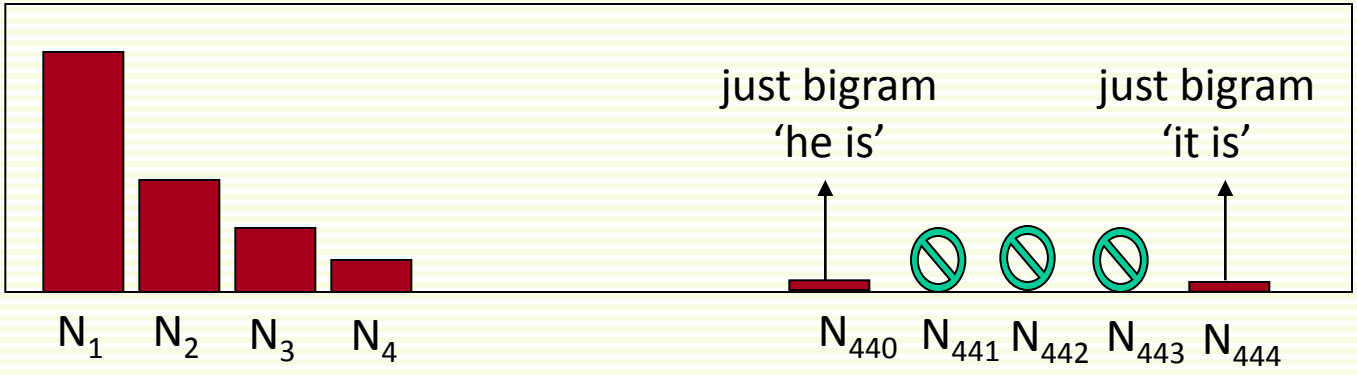
# Fixing Good Turing

- If n-gram x that occurs **r** times: $\mathbf{P_{GT}(x) = (r+1)\frac{N_{r+1}}{N \cdot N_r}}$
- Does not work well for high values of **r**
- $\mathbf{N_r}$ is not reliable estimate of the number of n-grams that occur with rate **r**
  - in particular, fails for the most frequent **r** since $\mathbf{N_{r+1}}$=0



# Fixing Good Turing

- MLE is reliable for higher values of **r**
- choose threshold **t**, say **t** = 6
  - best threshold depends on data
- for **r** ≥ t, use $\mathbf{P}_{MLE}(w_1 \ldots w_n) = \mathbf{C}(w_1 \ldots w_n)/\mathbf{N}$
- for **r** < t, use $P_{GT}$
  - if using GT for rate **r**, make sure $\mathbf{N_{r+1}} > 0$
  - also make sure $\frac{\mathbf{N}_{r+1}}{\mathbf{N}_r} < \frac{\mathbf{r}}{(\mathbf{r}+1)}$
  - otherwise the new rate $r^* = \frac{(r+1)N_{r+1}}{N_r}$ is larger than old rate **r**



# Smoothing: Fixing Good Turing

- Have to normalize our estimates  so that they add up to 1
- There are various ways to normalize
- We change only the observed ngrams
  - unobserved bigrams occupy the same space as GT says they should occupy
    - suppose the total space for unseen n-grams is 1/20
    - normalize the weight of the observed n-grams so that the total is 19/20

# Good-Turing (GT) Example

- $\mathbf{P}_{GT}$(n-gram occuring r times) = $(\mathbf{r}+1)\frac{\mathbf{N}_{r+1}}{\mathbf{N}\mathbf{N}_r}$
- Vocabulary is {a,b,c}
- Possible bigrams: {aa,ab,ba,bb,ac,bc,ca,cb,cc}
- Corpus: babaacbcacac
  - observed bigrams are {ba, ab, ba, aa, ac, cb, bc, ca, ac, ca, ac}
  - unobserved bigrams: bb,cc
- Observed bigram counts
  - ab: 1, aa: 1,cb: 1, bc: 1, ba: 2, ca: 2, ac: 3
- $\mathbf{N}_0$=2, $\mathbf{N}_1$=4, $\mathbf{N}_2$=2, $\mathbf{N}_3$=1, $\mathbf{N}$ = 12
- Threshold $\mathbf{t}$ = 3
  - GT up to and including $\mathbf{r}$ = 2, MLE starting with $\mathbf{r}$ = 3 and higher
- GT: $\mathbf{P}$(bb) = $\mathbf{P}$(cc)= (0+1)×($\mathbf{N}_1$/($\mathbf{N}$×$\mathbf{N}_0$))=4/(12×2) = 1/6
- GT: $\mathbf{P}$(ab) = $\mathbf{P}$(aa)=$\mathbf{P}$(cb)=$\mathbf{P}$(bc)= (1+1)×($\mathbf{N}_2$/($\mathbf{N}$×$\mathbf{N}_1$)) = 1/12
- GT: $\mathbf{P}$(ba) = $\mathbf{P}$(ca)= (2+1)×($\mathbf{N}_3$/($\mathbf{N}$×$\mathbf{N}_2$)) = 1/8
- MLE: $\mathbf{P}$(ac) = 3/12 = 1/4

# Good-Turing (GT) Example, Normalization

- Before normalization
  - Unseen bigrams
    - **P**’(bb) = **P**’(cc) = 1/6
  - Observed bigrams
    - **P**’(ab) = **P**’(aa) = **P**’(cb) = **P**’(bc) = 1/12
    - **P**’(ba) = **P**’(ca) = 1/8
    - **P**’(ac) = 1/4
  - **P**’(·) to indicate that the above are not true probabilities, they don’t add up to 1
- Unseen bigrams should occupy **P**’(bb) + **P**’(cc) = 1/3 of space after normalization
- Weight of observed bigrams should be 1 – 1/3 = 2/3
  - **P**’(ab) + **P**’(aa) + **P**’(cb) + **P**’(bc) + **P**’(ba) + **P**’(ca) + **P**’(ac) = 10/12 = 5/6
  - Solve for **y** equation:
    - (5/6) × **y** = 2/3
    - **y** = 4/5
  - Multiply **P**’(·) for the observed bigrams by 4/5
    - **P**(ab) = **P**(aa) = **P**(cb) = **P**(bc)= (1/12)×(4/5) = 1/15
    - **P**(ba) = **P**(ca) = (1/8)×(4/5) = 1/10
    - **P**(ac) = (1/4)×(4/5) =1/5

# Good-Turing (GT) Example

- Let's calculate **P**(abcab) using our model
- Probabilities, normalized
  - **P**(bb) = **P**(cc)= 1/6
  - **P**(ab) = **P**(aa) = **P**(cb) = **P**(bc)= 1/15
  - **P**(ba) = **P**(ca) = 1/10
  - **P**(ac) = 1/5
- Also need probabilities for unigrams a,b,c, compute with MLE
  - Corpus = "babaacbcacac"
  - **P**(a) = 5/12, **P**(b) = 3/12, **P**(c)=4/12
- Recall bigram approximation

$$\mathbf{P}(\text{abcab}) \approx \mathbf{P}(\text{a})\,\mathbf{P}(\text{b}|\text{a})\,\mathbf{P}(\text{c}|\text{b})\,\mathbf{P}(\text{a}|\text{c})\,\mathbf{P}(\text{b}|\text{a})$$

$$= \mathbf{P}(\text{a})\frac{\mathbf{P}(\text{ab})}{\mathbf{P}(\text{a})}\frac{\mathbf{P}(\text{bc})}{\mathbf{P}(\text{b})}\frac{\mathbf{P}(\text{ca})}{\mathbf{P}(\text{c})}\frac{\mathbf{P}(\text{ab})}{\mathbf{P}(\text{a})}$$

$$= \frac{5}{12}\frac{1/15}{5/12}\frac{1/15}{3/12}\frac{1/10}{4/12}\frac{1/15}{5/12} \approx 0.0008533$$

# Applications of LM: Language Identification

- Texts in the same language share similar characteristics
  - In English *"ing"* is more probable than in French
- Character based model
  - 26 letters (case insensitive)
  - dan also add punctuation
- Training phase
  - pre-classified documents (known language/author)
  - construct the language model for each document class separately
- Testing phase
  - compute probability of a sentence under English, Greec, etc. languages
  - assign to the language that gives maximum probability to the sentence

# Spam/Ham Classification

- Construct character based model for ham and separately for spam from training data
  - use all 256 characters
  - punctuation is important
- For new email, evaluate its character sequence using spam character model and ham character model
- Highest probability model wins