CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 3: Language Models
(Intro to Probability Models for NLP)

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Last lecture's key concepts

Dealing with words:

- Zipf's Law
- Tokenization, normalization

Morphology (word structure):

- stems, affixes
- Derivational vs. inflectional morphology
- Compounding
- Stem changes
- Morphological analysis and generation

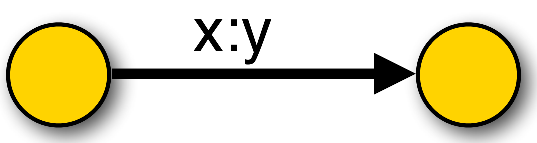
Finite-state methods in NLP

- Finite-state automata
- Finite-state transducers
- Composing finite-state transducers

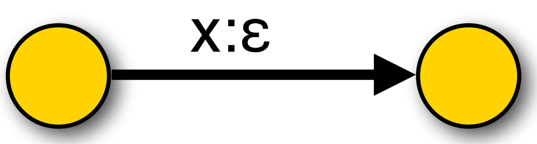
Finite-state transducers

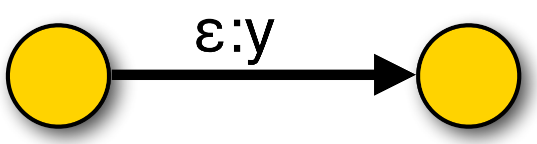
– FSTs define a **relation** between two regular languages.

– Each state transition maps (**transduces**) a character from the input language to a character (or a sequence of characters) in the output language



– By using the **empty character** (ε), characters can be **deleted** (x:ε) or **inserted**(ε:y)





– FSTs can be composed (**cascaded**), allowing us to define **intermediate representations.**

Today's lecture

How can we distinguish word salad, spelling errors and grammatical sentences?

- Language models define probability distributions over the strings in a language.
- N-gram models are the simplest and most common kind of language model.
- We'll look at how these models are defined, how to estimate (learn) them, and what their shortcomings are.

We'll also review some very basic probability theory.

Why do we need language models?

Many NLP tasks require **natural language output**:

- **Machine translation**: return text in the target language
- **Speech recognition**: return a transcript of what was spoken
- **Natural language generation**: return natural language text
- **Spell-checking**: return corrected spelling of input

Language models define **probability distributions over (natural language) strings or sentences**.

➔ We can use a language model to **score possible output strings** so that we can choose the best (i.e. most likely) one: if $P_{LM}(A) > P_{LM}(B)$, return A, not B

Hmmm, but…

… what does it mean for a language model to “define a probability distribution”?

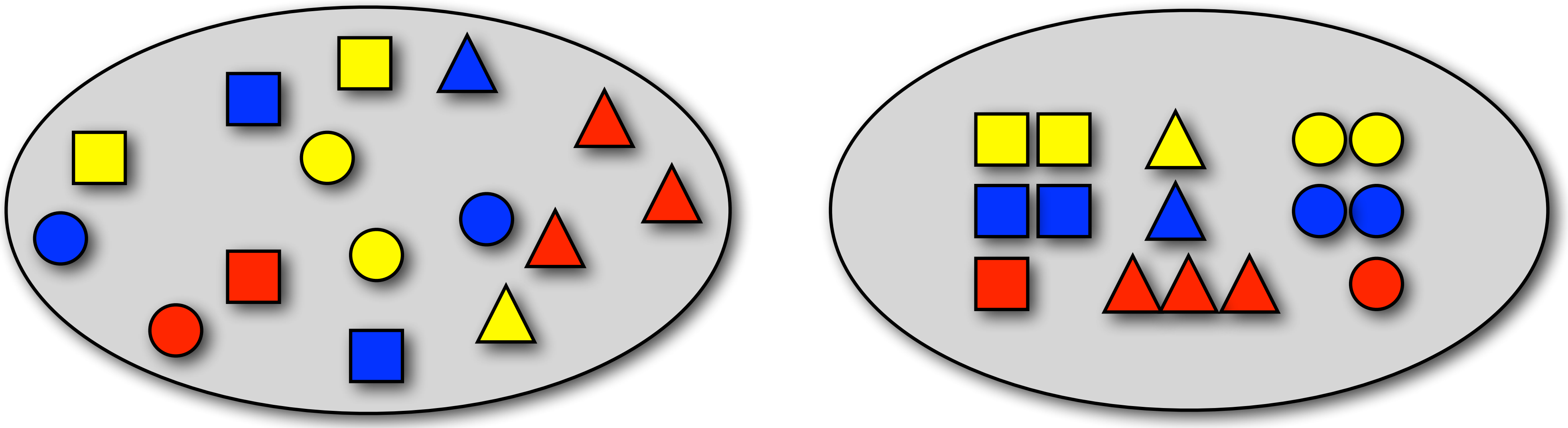
… *why* would we want to define probability distributions over languages?

… how can we construct a language model such that it *actually* defines a probability distribution?

Reminder: Basic Probability Theory

Sampling with replacement

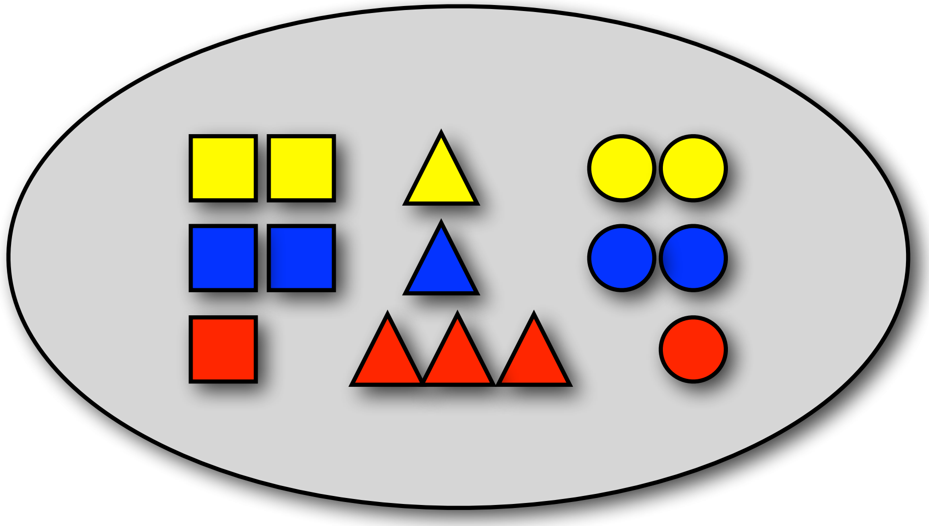
Pick a random shape, then put it back in the bag.



$P($🟦$) = 2/15$ | $P($🟥$) = 1/15$ | $P($🟥 or ▲$) = 2/15$
$P(\text{blue}) = 5/15$ | $P(\text{red}) = 5/15$ | $P(△ \mid \text{red}) = 3/5$
$P(\text{blue} \mid □) = 2/5$ | $P(□) = 5/15$

Sampling with replacement

Pick a random shape, then put it back in the bag.
What sequence of shapes will you draw?



P(●▲▲■)
= $1/15 \times 1/15 \times 1/15 \times 2/15$
= $2/50625$

P(▲●●▲)
= $3/15 \times 2/15 \times 2/15 \times 3/15$
= $36/50625$

P(■) = $2/15$
P(blue) = $5/15$
P(blue | □) = $2/5$

P(■) = $1/15$
P(red) = $5/15$
P(□) = $5/15$

P(■ or ▲) = $2/15$
P(△ | red) = $3/5$

Sampling 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?'

$P(\texttt{of}) = 3/66$	$P(\texttt{her}) = 2/66$
$P(\texttt{Alice}) = 2/66$	$P(\texttt{sister}) = 2/66$
$P(\texttt{was}) = 2/66$	$P(\texttt{,}) = 4/66$
$P(\texttt{to}) = 2/66$	$P(\texttt{'}) = 4/66$

Sampling with replacement

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

$P(\texttt{of}) = 3/66$	$P(\texttt{her}) = 2/66$
$P(\texttt{Alice}) = 2/66$	$P(\texttt{sister}) = 2/66$
$P(\texttt{was}) = 2/66$	$P(\texttt{,}) = 4/66$
$P(\texttt{to}) = 2/66$	$P(\texttt{'}) = 4/66$

In this model, *P(English sentence) = P(word salad)*

Probability theory: terminology

Trial (aka "experiment")

Picking a shape, predicting a word

Sample space Ω:

The **set of all possible outcomes**
(all shapes; all words in *Alice in Wonderland*)

Event $\omega \subseteq \Omega$:

An **actual outcome** (a subset of Ω)
(predicting '*the*', picking a triangle)

Random variable X: $\Omega \rightarrow \mathrm{T}$

A function from the sample space (often the identity function)
Provides a 'measurement of interest' from a trial/experiment
(Did we pick 'Alice'/a noun/a word starting with "x"/…?)

What is a probability distribution?

$P(\omega)$ defines a **distribution** over Ω iff

1) *Every* event ω has a probability $P(\omega)$ between 0 and 1:

$$0 \leq P(\omega \subseteq \Omega) \leq 1$$

2) The *null* event ∅ has probability $P(\varnothing) = 0$:

$$P(\emptyset) = 0$$

3) And the probability of all *disjoint* events sums to 1.

$$\sum_{\omega_i \subseteq \Omega} P(\omega_i) = 1 \text{ if } \forall j \neq i : \omega_i \cap \omega_j = \emptyset \\ \text{and } \bigcup_i \omega_i = \Omega$$

Discrete probability distributions: single trials

'Discrete': a fixed (often finite) number of outcomes

Bernoulli distribution (two possible outcomes)
Defined by the probability of success (= head/yes)
The probability of *head* is p. The probability of *tail* is $1-p$.

Categorical distribution (*N* possible outcomes $c_1 \ldots c_N$)
The probability of category/outcome c_i is p_i ($0 \leq p_i \leq 1$; $\sum_i p_i = 1$).

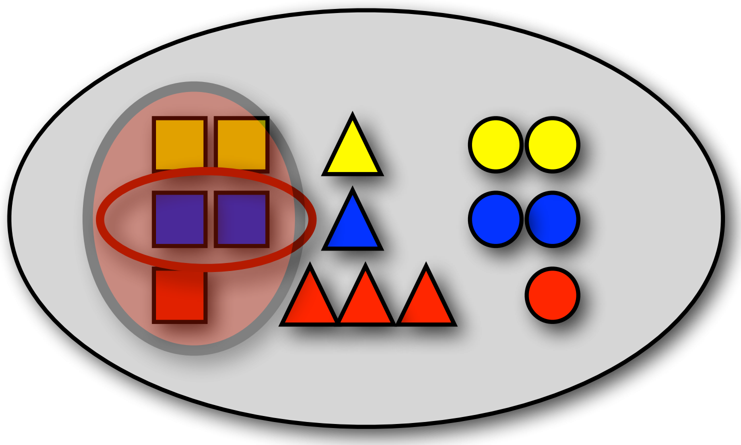
- e.g. the probability of getting a six when rolling a die once
- e.g. the probability of the next word (picked among a vocabulary of N words)

(NB: Most of the distributions we will see in this class are categorical.
Some people call them multinomial distributions, but those refer to *sequences* of trials, e.g. the probability of getting five sixes when rolling a die ten times)

Joint and Conditional Probability

The conditional probability of X given Y, $P(X \mid Y)$, is defined in terms of the probability of Y, $P(Y)$, and the joint probability of X and Y, $P(X,Y)$:

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



$P(\text{blue} \mid$ ■ $) = 2/5$

The chain rule

The joint probability $P(X,Y)$ can also be expressed in terms of the conditional probability $P(X \mid Y)$

$$P(X,Y) \quad = \quad P(X|Y)P(Y)$$

This leads to the so-called chain rule:

$$\begin{aligned} P(X_1, X_2, \ldots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_2, X_1)....P(X_n|X_1, ...X_{n-1}) \\ &= P(X_1)\prod_{i=2}^{n} P(X_i|X_1 \ldots X_{i-1}) \end{aligned}$$

Independence

Two random variables X and Y are independent if

$$P(X,Y) = P(X)P(Y)$$

If X and Y are independent, then $P(X \mid Y) = P(X)$:

$$\begin{aligned} P(X|Y) &= \frac{P(X,Y)}{P(Y)} \\ &= \frac{P(X)P(Y)}{P(Y)} \quad (X,\ Y \text{ independent}) \\ &= P(X) \end{aligned}$$

Probability models

Building a probability model consists of two steps:
1. Defining the model
2. Estimating the model's parameters
   (= training/learning)

Models (almost) always make
independence *assumptions.*

That is, even though X and Y are not actually independent, our model may treat them as independent.

This reduces the number of model parameters that we need to estimate (e.g. from n^2 to $2n$)

Language modeling with n-grams

Language modeling with N-grams

A language model over a vocabulary V
assigns probabilities to strings drawn from V^*.

Recall the **chain rule**:

$$P(w^{(1)} \ldots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} \mid w^{(1)}) \cdot \ldots \cdot P(w^{(i)} \mid w^{(i-1)}, \ldots, w^{(1)})$$

An **n-gram** language model assumes each word
depends only on **the last n−1 words**:

$$P_{ngram}(w^{(1)} \ldots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} \mid w^{(1)}) \cdot \ldots \cdot P(w^{(i)} \mid w^{(i-1)}, \ldots, w^{(1-(n+1))})$$

N-gram models

N-gram models *assume* each word (event) depends only on the previous n–1 words (events):

Unigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$

Bigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$

Trigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)})$

Such independence assumptions are called Markov assumptions (of order n–1).

A unigram model for Alice

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

$P(\texttt{of}) = 3/66$	$P(\texttt{her}) = 2/66$
$P(\texttt{Alice}) = 2/66$	$P(\texttt{sister}) = 2/66$
$P(\texttt{was}) = 2/66$	$P(\texttt{,}) = 4/66$
$P(\texttt{to}) = 2/66$	$P(\texttt{'}) = 4/66$

In this model, $P(\textit{English sentence}) = P(\textit{word salad})$

A bigram model for Alice

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?'

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{tired}) = 1$

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{use}) = 1$

$P(w^{(i)} = \textbf{sister} \mid w^{(i-1)} = \textbf{her}) = 1$

$P(w^{(i)} = \textbf{beginning} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{reading} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{bank} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{book} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{use} \mid w^{(i-1)} = \textbf{the}) = 1/3$

Using a bigram model for Alice

English

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?'

Word Salad

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

Now, $P(\textit{English}) \gg P(\textit{word salad})$

$P(w^{(i)} = \texttt{of} \mid w^{(i-1)} = \texttt{tired}) = 1$

$P(w^{(i)} = \texttt{of} \mid w^{(i-1)} = \texttt{use}) = 1$

$P(w^{(i)} = \texttt{sister} \mid w^{(i-1)} = \texttt{her}) = 1$

$P(w^{(i)} = \texttt{beginning} \mid w^{(i-1)} = \texttt{was}) = 1/2$

$P(w^{(i)} = \texttt{reading} \mid w^{(i-1)} = \texttt{was}) = 1/2$

$P(w^{(i)} = \texttt{bank} \mid w^{(i-1)} = \texttt{the}) = 1/3$

$P(w^{(i)} = \texttt{book} \mid w^{(i-1)} = \texttt{the}) = 1/3$

$P(w^{(i)} = \texttt{use} \mid w^{(i-1)} = \texttt{the}) = 1/3$

Where do we get the probabilities from?

Learning (estimating) a language model

Where do we get the parameters of our model (its actual probabilities) from?

$$P(w^{(i)} = \text{‘}the\text{’} \mid w^{(i-1)} = \text{‘}on\text{’}) = ???$$

We need (a large amount of) text as training data to estimate the parameters of a language model.

The most basic parameter estimation technique: relative frequency estimation (= counts)

$$P(w^{(i)} = \text{‘}the\text{’} \mid w^{(i-1)} = \text{‘}on\text{’}) = C(\text{‘}on\ the\text{’}) \,/\, C(\text{‘}on\text{’})$$

Also called Maximum Likelihood Estimation (MLE)

NB: MLE assigns *all* probability mass to events that occur in the training corpus.

Are n-gram models actual language models?

How do n-gram models define P(L)?

An n-gram model defines $P_{ngram}(w^{(1)} \dots w^{(N)})$ in terms of the probability of predicting each word: $P_{bigram}(w^{(1)} \dots w^{(N)}) = \prod_{i=1\dots N} P(w^{(i)} \mid w^{(i-1)})$

With a fixed vocabulary V, it's easy to make sure $P(w^{(i)} \mid w^{(i-1)})$ is a distribution: $\sum_{i=1\dots|V|} P(w_i \mid w_j) = 1$ and $\forall_{i,j} 0 \leq P(w_i \mid w_j) \leq 1$

If $P(w^{(i)} \mid w^{(i-1)})$ is a distribution, this model defines *one* distribution (over all strings) *for each length* N

But the strings of a language L *don't all have the same length*

English = {*"yes!", "I agree", "I see you", …*}

And there is *no* N_{max} that limits how long strings in L can get.

Solution: the **EOS (end-of-sentence)** token!

How do n-gram models define P(L)?

Think of a language model as a stochastic process:

- At each time step, randomly pick one more word.
- Stop generating more words when the word you pick is a special end-of-sentence (EOS) token.

To be able to pick the EOS token, we have to modify our training data so that each sentence ends in EOS.

This means our vocabulary is now $V^{EOS} = V \cup \{EOS\}$

We then get an actual language model,
i.e. a distribution over strings of *any* length

Technically, this is only true because P(EOS | …) will be high enough that we are always guaranteed to stop after having generated a finite number of words

Why do we care about having one model for all lengths?
We can now compare the probabilities of strings of different lengths, because they're computed by the same distribution.

A couple more tweaks…

Handling unknown words: UNK

Training:

- Assume a fixed vocabulary (e.g. all words that occur at least *n* times in the training corpus)
- Replace all other words in the corpus by a token <UNK>
- Estimate the model on this modified training corpus.

Testing (e.g to compute probability of a string):

- Replace any words not in the vocabulary by <UNK>

Refinements:

use different UNK tokens for different types of words (numbers, etc.).

What about the beginning of the sentence?

In a trigram model

$$P(w^{(1)}w^{(2)}w^{(3)}) = P(w^{(1)})P(w^{(2)} \mid w^{(1)})P(w^{(3)} \mid w^{(2)}, w^{(1)})$$

only the third term $P(w^{(3)} \mid w^{(2)}, w^{(1)})$ is an actual trigram probability. What about $P(w^{(1)})$ and $P(w^{(2)} \mid w^{(1)})$?

If this bothers you:

Add n–1 **beginning-of-sentence** (BOS) symbols to each sentence for an n–gram model:

```
BOS₁ BOS₂ Alice was …
```

Now the unigram and bigram probabilities involve only BOS symbols.

To recap…

Estimating a bigram models with BOS (<s>), EOS (</s>) and UNK using MLE

1. Replace all rare words in training corpus with UNK
2. Bracket each sentence by special start and end symbols:

```
<s> Alice was beginning to get very tired… </s>
```

3. Vocabulary V' = all tokens in modified training corpus
   (all common words, UNK, <s>, </s>)
4. Count the frequency of each bigram….

$$C(\text{<s> Alice}) = 1,\ C(\text{Alice was}) = 1,\ \ldots$$

5. .... and normalize these frequencies to get probabilities:

$$P(\textit{was} \mid \textit{Alice}) = \sum_{w_i \in V'} \frac{C(\textit{Alice was})}{C(\textit{Alice } w_i)}$$

Using language models

How do we use language models?

Independently of any application, we can use a language model as a random sentence generator
(i.e we sample sentences according to their language model probability)

Systems for applications such as machine translation, speech recognition, spell-checking, generation, often produce multiple candidate sentences as output.

- We prefer output sentences S_{Out} that have a higher probability
- We can use a language model $P(S_{Out})$ to score and rank these different candidate output sentences, e.g. as follows:

$$\text{argmax}_{S_{Out}} P(S_{Out} \mid \text{Input}) = \text{argmax}_{S_{Out}} P(\text{Input} \mid S_{Out}) P(S_{Out})$$

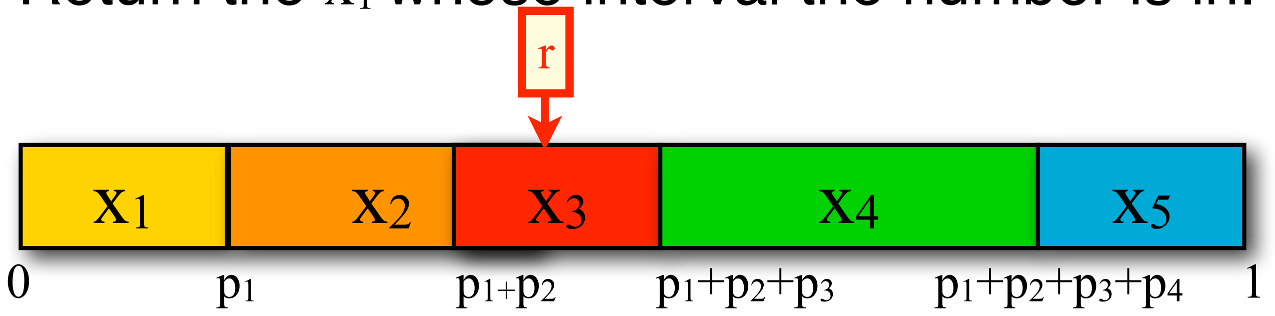
Using n-gram models to generate language

Generating from a distribution

How do you generate text from an *n*-gram model?

That is, how do you sample from a distribution $P(X \mid Y=y)$?

- Assume X has N possible outcomes (values): $\{x_1, \ldots, x_N\}$ and $P(X=x_i \mid Y=y) = p_i$
- Divide the interval [0,1] into N smaller intervals according to the probabilities of the outcomes
- Generate a random number r between 0 and 1.
- Return the x_1 whose interval the number is in.



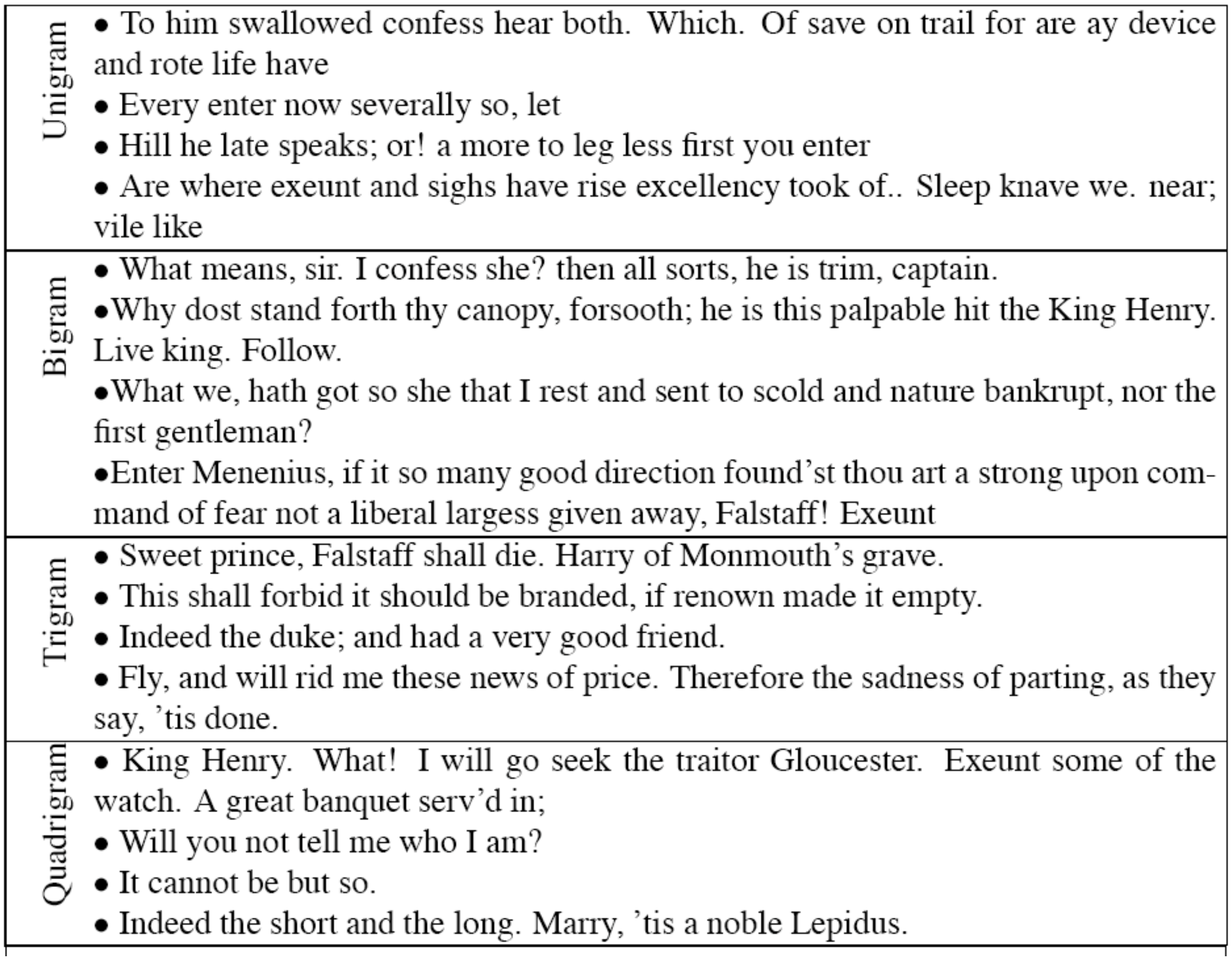
Generating the Wall Street Journal

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 planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Generating Shakespeare



Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as corpus

The Shakespeare corpus consists of N=884,647 word **tokens** and a vocabulary of V=29,066 word **types**

Shakespeare produced 300,000 bigram types
out of V^2= 844 million possible bigram types.

99.96% of possible bigrams don't occur in the corpus.

Our relative frequency estimate assigns non-zero probability to only 0.04% of the possible bigrams

That percentage is even lower for trigrams, 4-grams, etc.

4-grams *look* like Shakespeare because they *are* Shakespeare!

MLE doesn't capture unseen events

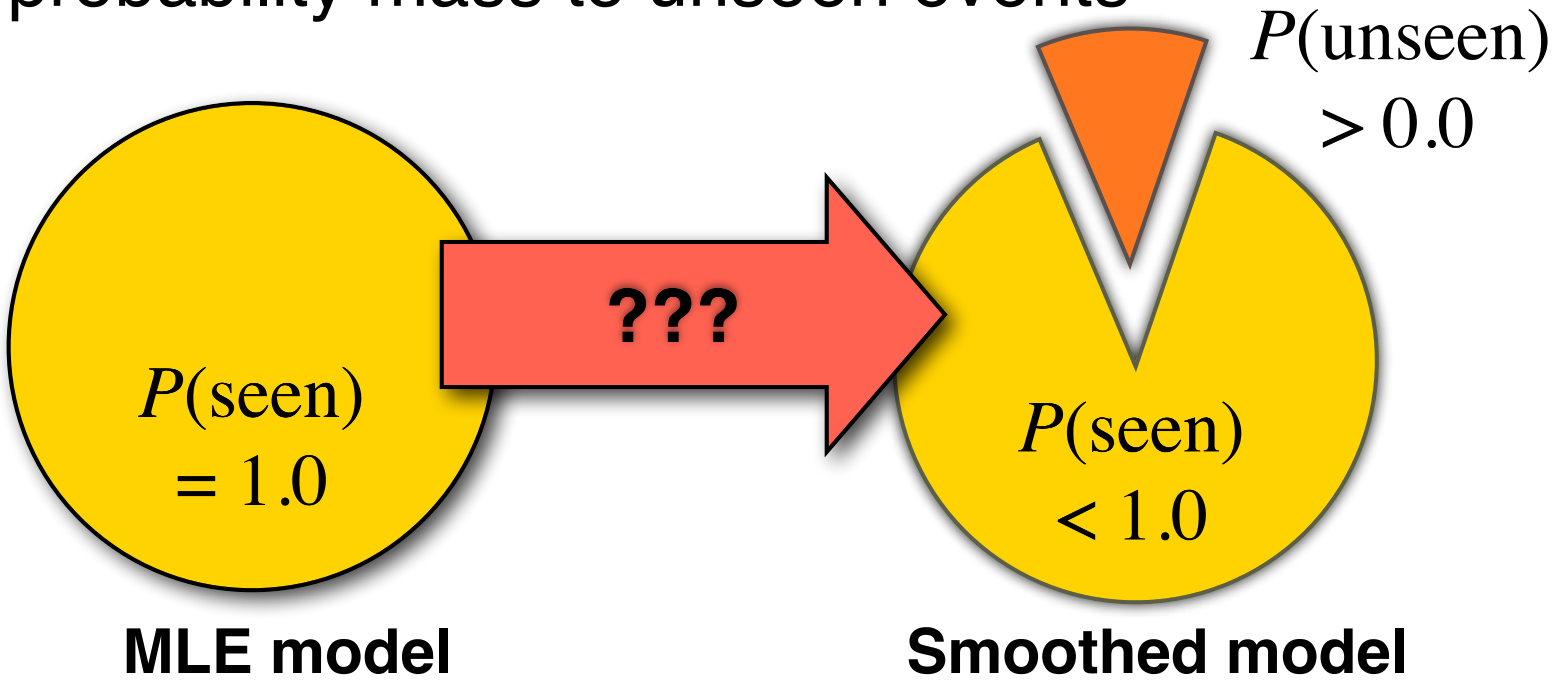
We estimated a model on 440K word tokens, but:

Only 30,000 word types occur in the training data
Any word that does not occur in the training data
has zero probability!

Only 0.04% of all possible bigrams (over 30K word types) occur in the training data
Any bigram that does not occur in the training data
has *zero* probability (even if we have seen both words in the bigram)

How we assign non-zero probability to unseen events?

We have to “smooth” our distributions to assign some probability mass to unseen events



We won’t talk much about smoothing this year.

Smoothing methods

Add-one smoothing:
Hallucinate counts that didn't occur in the data

Linear interpolation:
$$\tilde{P}(w \mid w', w'') = \lambda \hat{P}(w \mid w', w'') + (1 - \lambda)\tilde{P}(w \mid w')$$
Interpolate n-gram model with (n–1)-gram model.

Absolute Discounting: Subtract constant count from frequent events and add it to rare events

Kneser-Ney: AD with modified unigram probabilities

Good-Turing: Use probability of rare events to estimate probability of unseen events

Add-One (Laplace) Smoothing

A really simple way to do smoothing:
Increment the actual observed count of every *possible* event (e.g. bigram) by a hallucinated count of 1
(or by a hallucinated count of some k with $0<k<1$).

Shakespeare bigram model (roughly):

0.88 million actual bigram counts
+ 844.xx million hallucinated bigram counts

Oops. Now almost none of the counts in our model come from actual data. We're back to word salad.

K needs to be really small. But it turns out that that still doesn't work very well.

Evaluation

Intrinsic vs Extrinsic Evaluation

How do we know whether one language model is better than another?

There are two ways to evaluate models:

- intrinsic evaluation captures how well the model captures what it is supposed to capture (e.g. probabilities)
- extrinsic (task-based) evaluation captures how useful the model is in a particular task.

Both cases require an evaluation metric that allows us to measure and compare the performance of different models.

Intrinsic Evaluation of Language Models: Perplexity

Perplexity

The perplexity of a language models is defined as the inverse ($\frac{1}{P(\ldots)}$) of the probability of the test set, normalized ($\sqrt[N]{\ldots}$) by the # of tokens (N) in the test set.

If a LM assigns probability $P(w_1, \ldots, w_N)$ to a test corpus $w_1 \ldots w_N$, the LM's perplexity, $PP(w_1 \ldots w_N)$, is

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

A LM with lower perplexity is better because it assigns a higher probability to the unseen test corpus.

LM_1 and LM_2's perplexity can only be compared if they use the same vocabulary

— Trigram models have lower perplexity than bigram models;

— Bigram models have lower perplexity than unigram models, etc.

Practical issues

Since language model probabilities are very small, multiplying them together often yields to underflow.

It is often better to use logarithms instead, so replace

$$PP(w_1 \dots w_N) =_{def} \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1}, \dots, w_{i-n+1})}}$$

with

$$PP(w_1 \dots w_N) =_{def} \exp\left(-\frac{1}{N} \sum_{i=1}^{N} \log P(w_i | w_{i-1}, \dots, w_{i-n+1} \right)$$

Extrinsic (Task-Based) Evaluation of LMs: Word Error Rate

Intrinsic vs. Extrinsic Evaluation

Perplexity tells us which LM assigns a higher probability to unseen text

This doesn't necessarily tell us which LM is better for our task (i.e. is better at scoring candidate sentences)

Task-based evaluation:

- Train model A, plug it into your system for performing task T
- Evaluate performance of system A *on task T*.
- Train model B, plug it in, evaluate system B on same task T.
- Compare scores of system A and system B on task T.

Word Error Rate (WER)

Originally developed for speech recognition.

How much does the *predicted* sequence of words differ from the *actual* sequence of words in the correct transcript?

$$\text{WER} = \frac{\text{Insertions} + \text{Deletions} + \text{Substitutions}}{\text{Actual words in transcript}}$$

Insertions: "eat lunch" → "eat **a** lunch"

Deletions: "see **a** movie" → "see movie"

Substitutions: "drink **ice** tea"→ "drink **nice** tea"

To recap….

Today's key concepts

N-gram language models

- Independence assumptions
- Getting from n-grams to a distribution over a language
- Relative frequency (maximum likelihood) estimation
- Smoothing
- Intrinsic evaluation: Perplexity,
- Extrinsic evaluation: WER

Today's reading:

- Chapter 3 (3rd Edition)

Next lecture: Basic intro to machine learning for NLP