CE4045 CZ4045 SC4002
Natural Language Processing

N-gram Language Models

Dr. Sun Aixin

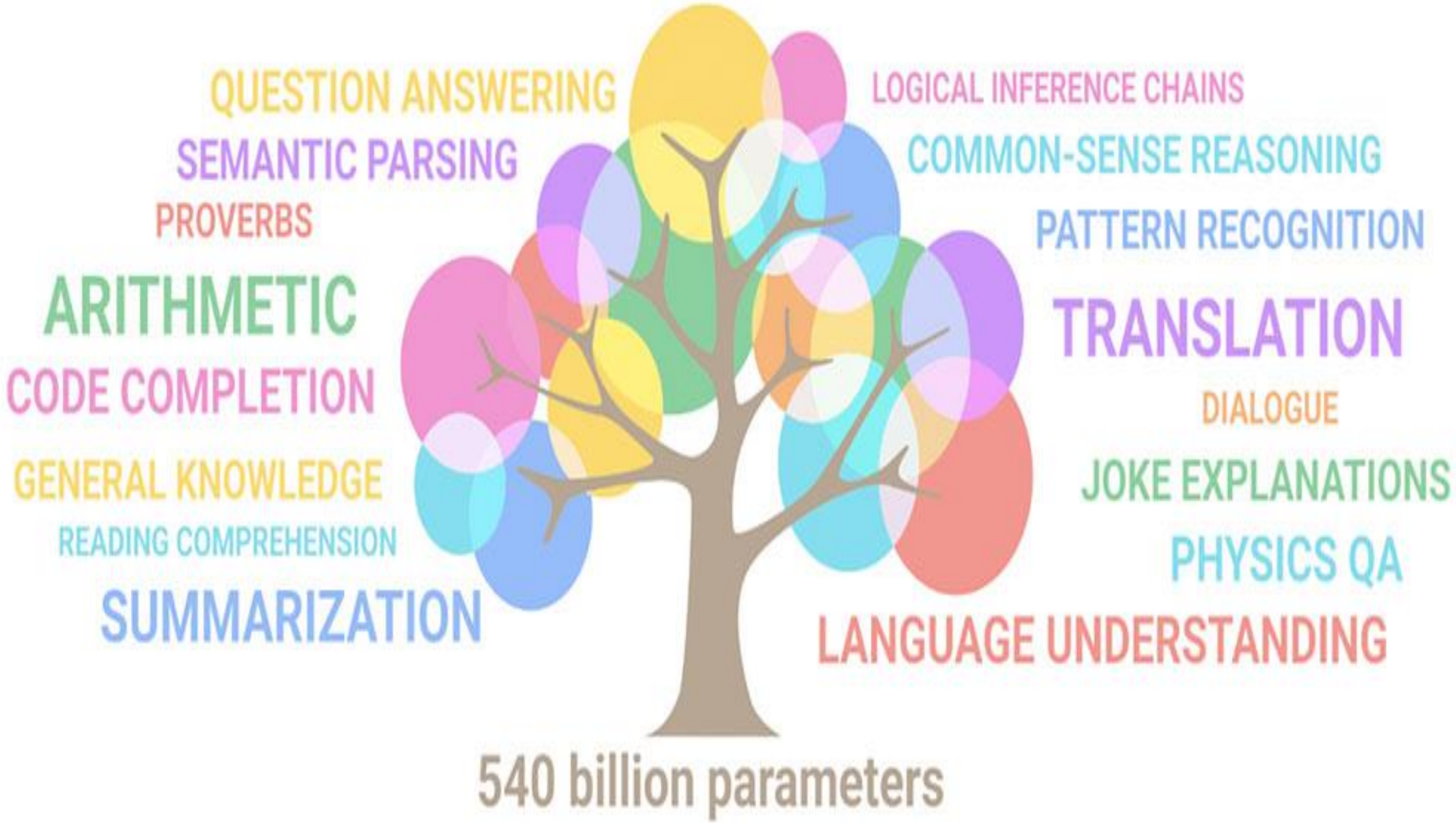
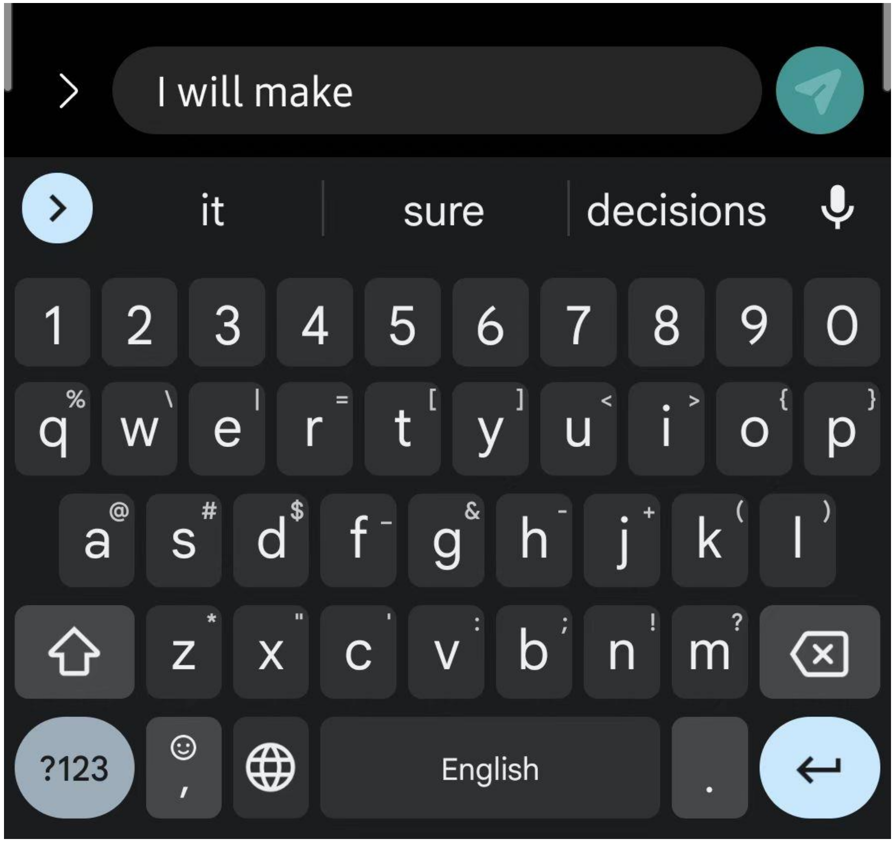


Image source: https://www.topbots.com/leading-nlp-language-models-2020/

Language Model: probabilities of sequences of words



I will make

- Let's play a game
  - Unlock your smart phone
  - Compose a message or a note
  - Start with word "I", the always select the first suggested word.
  - What is the sentence you get?
- Example:
  - "I will make ______
  - Suggested next word:
    - it
    - sure
    - decisions

Why Language Model?

- Which sequence is more like a proper sentence?
  - "*all of a sudden I notice three guys standing on the sidewalk*"
  - "*on guys all I of notice sidewalk three a sudden standing the*"

- Are such probabilities useful?
  - **Speech recognition**
    - "*I will be back soonish*" vs "*I will be bassoon dish*"
  - **Spelling correction** or **grammatical error prediction**
    - "Their are two midterms" vs "There are …"
    - "Everything has improve" vs "has improved"
  - **Machine translation**
    - "他(he) 向(to) 记者(reporters) 介绍了(introduced) 主要(main)内容 (content)
    - "he introduced reporters to the main contents of the statement"
    - "he briefed to reporters the main contents of the statement"
    - "he briefed reporters on the main contents of the statement"

N-gram Model: The simplest language model

- Language models
  - N-gram language models
  - Neural language models
  - Pre-trained language models
  - Multimodal language models (text, vision, sound…)

- N-gram model
  - An n-gram is a sequence of *n* words;
    - 2-gram called bigram, 3-gram called trigram
  - Word sequence: "I like natural language processing"
    - Bigram: "I like" "like natural" "natural language" "language processing".
    - Trigram: "I like natural", "like natural language" "natural language processing"
  - An important foundational tool for understanding the fundamental concepts in LM

Our task: computing $\boldsymbol{P(w|h)}$

- $\boldsymbol{P(w|h)}$: the probability of word w given some history h
  - Example : h is "I will make", and the word w is "it"
  - $P(w|h) = P(it|I\ will\ make)$

  - Estimate the probability $P(w|h)$ from a large text collection
    - Count number of times "I will make" appears
    - Count number of times "I will make it" appears

$$P(w|h) = \frac{C(I\ will\ make\ it)}{C(I\ will\ make)}$$

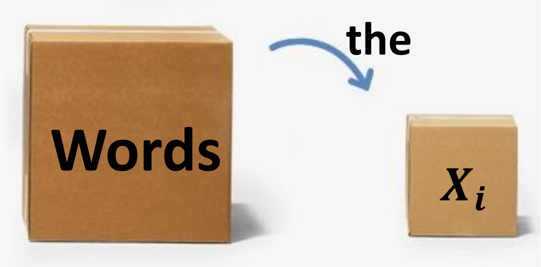
  - What if the history h is a long sequence like:
    "*<u>all of a sudden I notice three guys standing on the sidewalk</u>*"

$$\underbrace{\text{all of a sudden I notice three guys standing on the}}_{h}\ \underbrace{\text{sidewalk}}_{w}$$

Language is creative!

Chain rule of probability: a better way to compute $P(w|h)$

- Notations:
  - The probability of a particular random variable X_i taking on the value "the" is $P(X_i = \text{"}the\text{"})$, or simply $P(the)$
  - A sequence of N words either as $w_1, w_2, \ldots, w_n$ or $w_{1:n}$



  - The joint probability of each word in a sequence having a particular value $P(X_1 = w_1; X_2 = w_2; X_3 = w_3; \ldots; X_n = w_n)$ is $P(w_1, w_2, \ldots, w_n)$ or $P(w_{1:n})$

- Chain rule of probability
  - $\boldsymbol{P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1X_2)\ldots P(X_n|X_{1:n-1})}$
  - Applying the chain rule to words
  - $P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\ldots P(w_n|w_{1:n-1}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1})$
  - We could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities

N-gram: an approximation

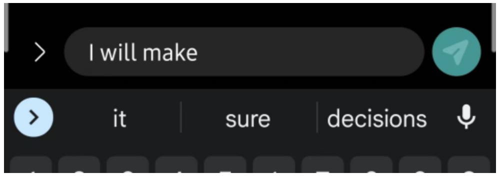
- We now have: $P(w_{1:n}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1})$
  - We could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities
  - Question is: **how to compute $\boldsymbol{P(w_n|w_{1:n-1})}$**?
  - In fact, we are at the starting point, to compute $\boldsymbol{P(w|h)}$: the probability of word w given some history h and $h = w_{1:n-1}$
- Bigram model
  - Approximates the probability of a word given all the previous words $\boldsymbol{P(w_n|w_{1:n-1})}$ by using only the conditional probability of the preceding word $\boldsymbol{P(w_n|w_{n-1})}$.
  - $\boldsymbol{P(w_n|w_{1:n-1})} \approx \boldsymbol{P(w_n|w_{n-1})}$

This is called Markov assumption

- Generalize to N-gram model ($N \geq 2$):

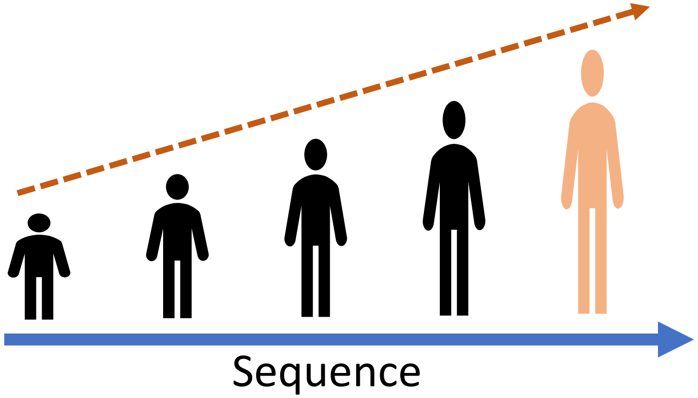
$$\boldsymbol{P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})}$$

Bigram example


I will make

it | sure | decisions

- 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 $P(w_n|w_{n-1})$.



- Before using bigram:

$$P(I\ will\ make\ it) = P(I) \times P(will|I) \times P(make|I\ will) \times P(it|I\ will\ make)$$

- With bigram

$$P(I\ will\ make\ it) = P(I) \times P(will|I) \times P(make|will) \times P(it|make)$$

Bigram model

- With bigram model $\boldsymbol{P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})}$
  - Our example

$$P(w|h) = P(it|I\ will\ make) \approx \boldsymbol{P(it|make)}$$

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

- Now, how to compute $\boldsymbol{P(w_n|w_{n-1})}$, like $\boldsymbol{P(it|make)}$?
  - Estimate bigram probabilities by **maximum likelihood estimation** or MLE
  - We estimate $P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$ where $C(\cdot)$ is the count, or frequency

make decisions.
make sure …
make it right
make it happen
make toys.

$C(make) = 5$
$C(make\ it) = 2$

$P(it|make) = 0.4$

An example with a mini-corpus of three sentences

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

- <s> is a special symbol at the beginning of the sentence to give us the bigram context of the first word.
- </s> is the end-symbol to make the bigram grammar a true probability distribution.

- Some of bigram probabilities from this corpus
  - $P(I| < s >) = \frac{2}{3} = 0.67$
  - $P(Sam| < s >) = \frac{1}{3} = 0.33$
  - $P(am|I) = \frac{2}{3} = 0.67$
  - $P(</s > |Sam) = \frac{1}{2} = 0.5$
  - $P(Sam|am) = \frac{1}{2} = 0.5$
  - $P(do|I) = \frac{1}{3} = 0.33$

The chance a sentence starts with I is 67%
The chance a sentence starts with Sam is 33%
The chance word am follows I is 67%

- In practice, trigram is more commonly used.
- If trigram is used, then we need to add extra context, e.g., $P(The| < s >< s >)$

Let’s take a larger example: Berkeley Restaurant Project

- A dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California.
- A sample of 9332 sentences is on the website http://www1.icsi.berkeley.edu/Speech/berp.html

- Some sample sentences:
  - 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
  - when is caffe venezia open during the day

Unigram counts and bigram counts for example words

- Number of sentences: 9332
- Number of unique words (vocabulary): 1446
- A **sample** of 7 words with unigram counts and bigram counts
- In this example, we do not explicitly show <s> and </s>

(I want) occurred 827 times

(Chinese food) occurred 82 times

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

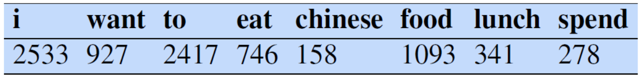
w_n

w_{n-1}

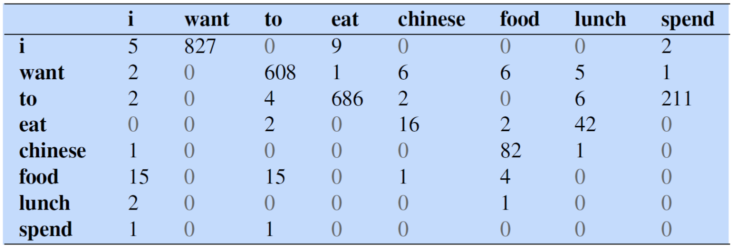
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram probabilities

- Based on the unigram counts and bigram counts
- We have the following sample bigram probabilities
- We can estimate other bigram probabilities as well

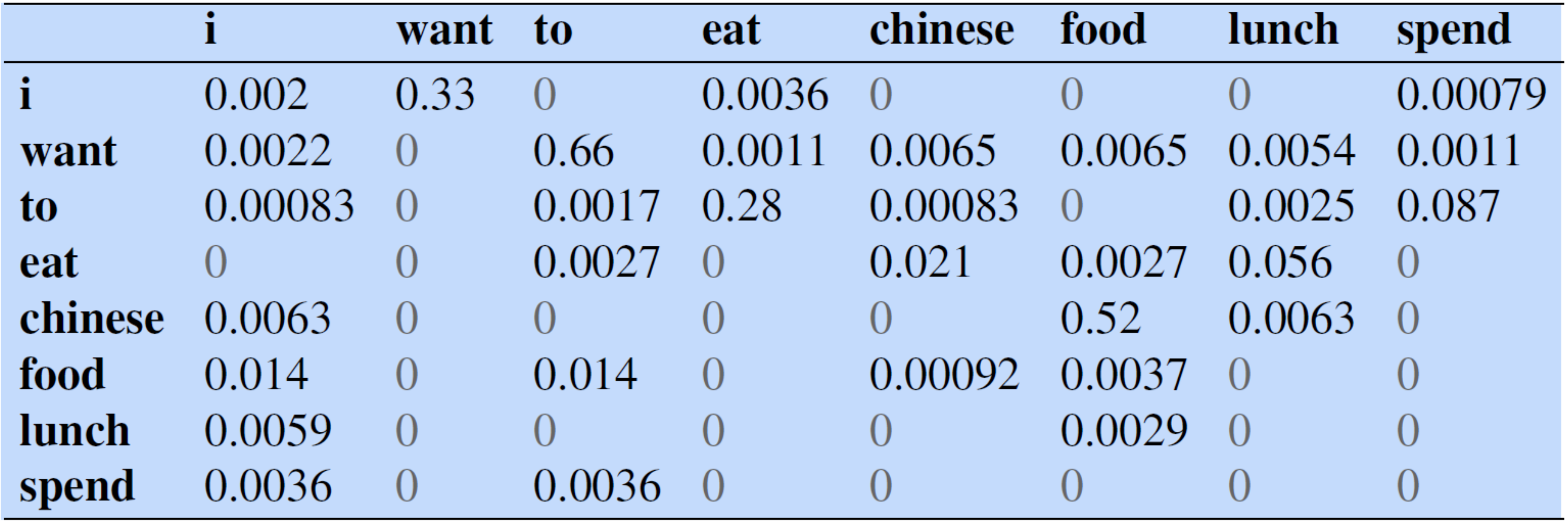


i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278



	i	want	to	eat	chinese	food	lunch	spend
i	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

$P(want|i) = 0.33$ $\quad$ $P(to|want) = 0.66$



	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

Now you can compute a probability of a sentence

	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	
spend	0.0036	0	0.0036	0	0	0	0	

Additional probabilities:
$P(i| < s >) = 0.25$
$P(english|want) = 0.0011$
$P(food|English) = 0.5$
$P(</s > |food) = 0.68$

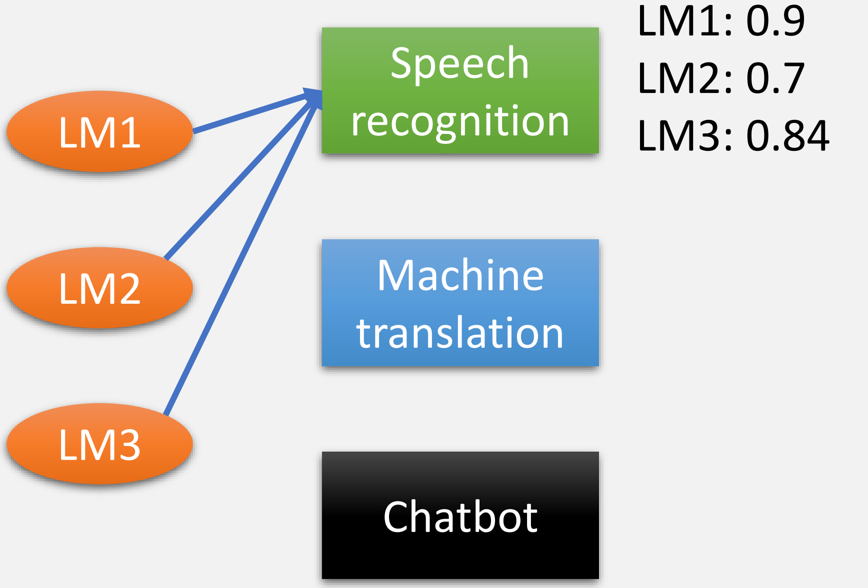
- Probability of sentence: "I want English food"
- P(<s> I want English food </s>)
  - =P(i|<s>) P(want|i) P(English|want) P(food|English) P(</s>|food)
  - =0.25x0.33x0.0011x0.5x0.68
  - =0.000031

Practical issues: Probability of a sentence is typically very small. To avoid numerical underflow, we use log probabilities. $\boldsymbol{p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)}$

Evaluating Language Model

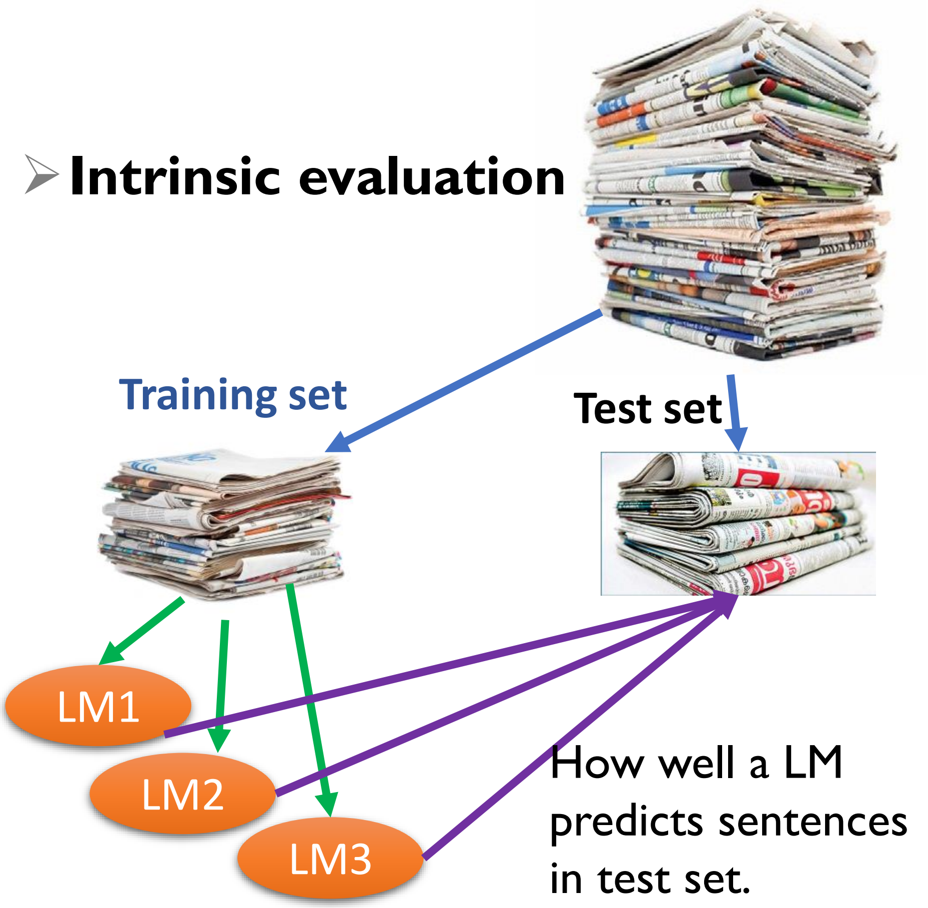
- We may learn a bigram model, a trigram model, or other types of LMs
- How do we know any model is good?

- **Extrinsic evaluation**



LM1: 0.9
LM2: 0.7
LM3: 0.84

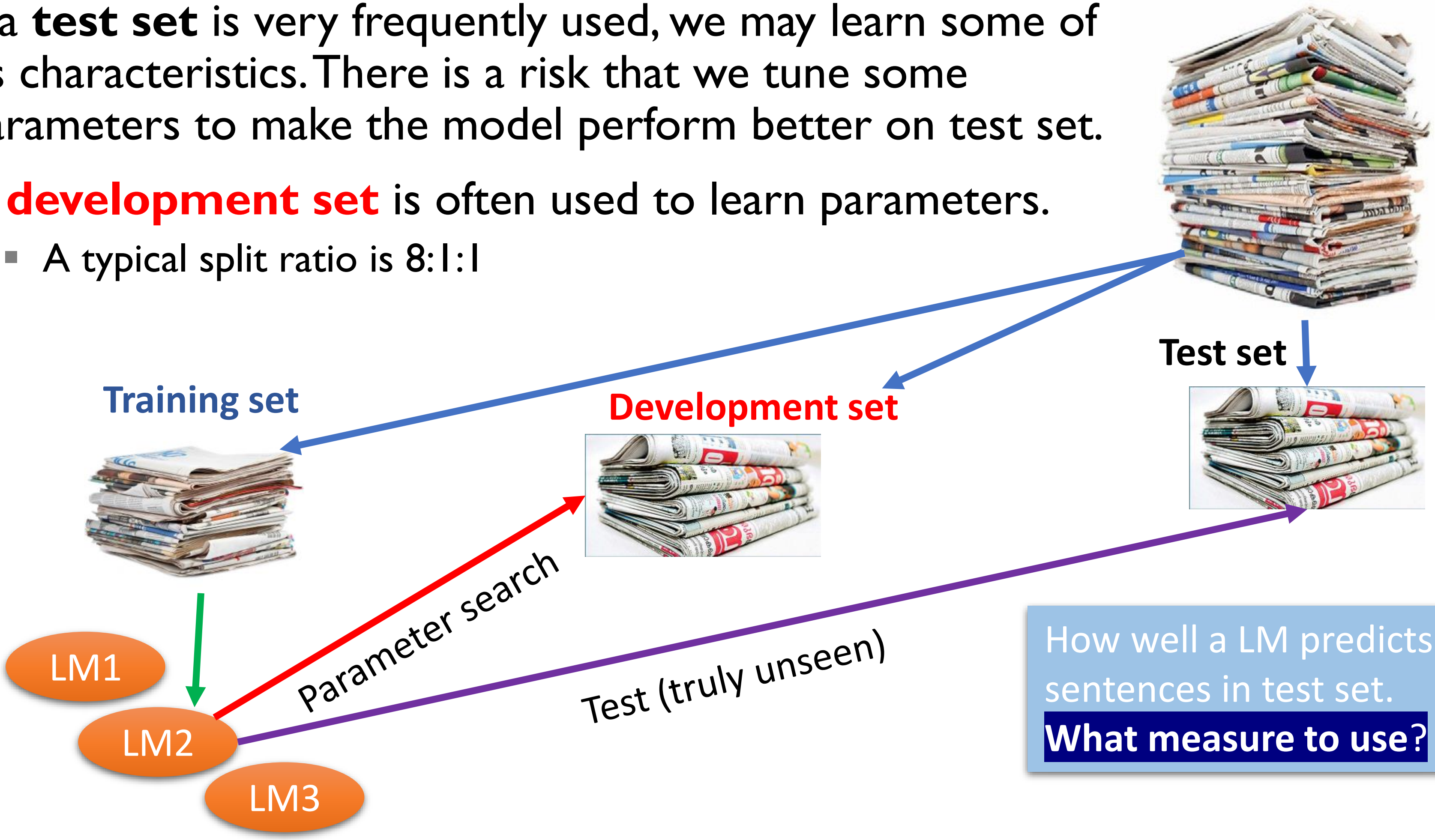
- **Intrinsic evaluation**



How well a LM predicts sentences in test set.

Intrinsic evaluation

- If a **test set** is very frequently used, we may learn some of its characteristics. There is a risk that we tune some parameters to make the model perform better on test set.
- A **development set** is often used to learn parameters.
  - A typical split ratio is 8:1:1



How well a LM predicts sentences in test set.
What measure to use?

Perplexity

- **Perplexity** (PP) is the probability of the test set (assigned by the language model), normalized by the number of words. N denotes number of words in a test data, $w1, \ldots, w_N$ is the test data.

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

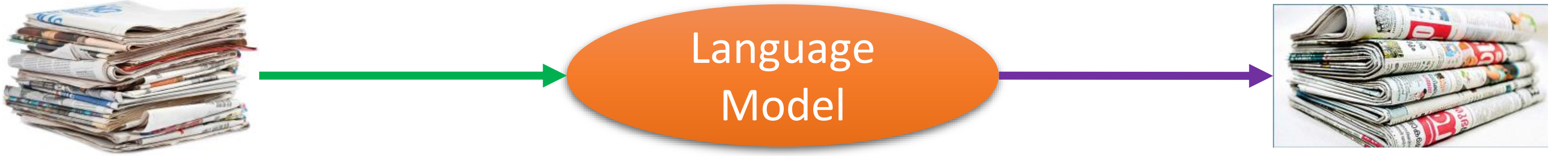
- Chain rule: $\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$

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

A good model gives high probability on test data, hence a **low perplexity** value.

An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance on a real task.

Let’s take a closer look at testing



Training set (Straits Times, Jan – Sept)

Test set (Straits Times, Oct – Dec)

Language Model

“Coronavirus disease **COVID-19** is ….”

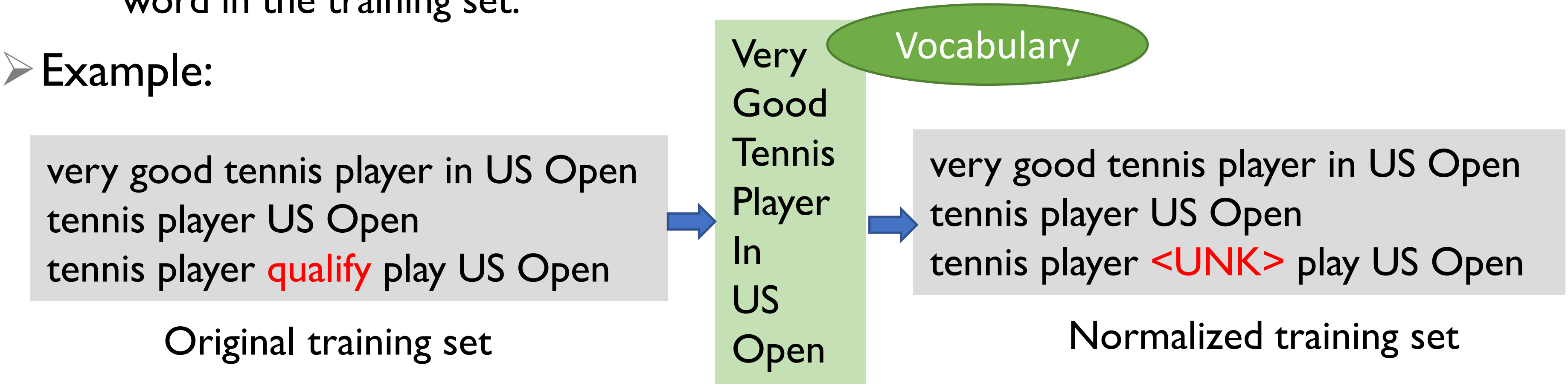
- The word “COVID-19” never appears in training data
- The model has no knowledge about this word, P(COVID-19|any_word) = 0
- Then the perplexity is undefined

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

- Happens to all unknown words, a.k.a **out of vocabulary (OOV)** words

Need to handle potentially unknown words in training

- In training, we add a pseudo-word called <UNK>.
  - Any potential unknown word in the test set is considered as an instance of <UNK>
- But where are the instances of <UNK> in training data?
- Modeling <UNK>
  - Choose a vocabulary (word list) that is fixed in advance, before training.
  - Convert in the training set any word that is not in this vocabulary to the unknown word token <UNK>, in a text normalization step.
  - Estimate the probabilities for <UNK> from its counts, just like any other regular word in the training set.
- Example:



very good tennis player in US Open
tennis player US Open
tennis player qualify play US Open

Original training set

Vocabulary

Very
Good
Tennis
Player
In
US
Open

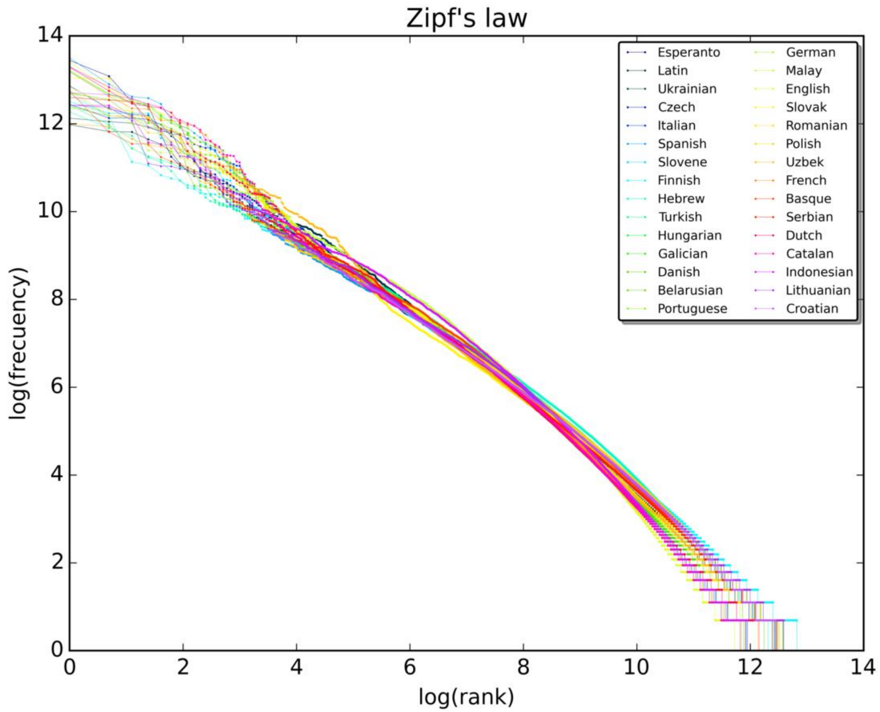
very good tennis player in US Open
tennis player US Open
tennis player <UNK> play US Open

Normalized training set

Modeling <UNK>

- How to create a vocabulary
  - Approach I: Based on some existing knowledge about the dataset, and fix a vocabulary in advance,
  - Approach II: Implicitly define a vocabulary based on word distribution

- Example for Approach II.
  - Count the frequency of every words in training data
  - All words that appear fewer than K times are considered as <UNK>, e.g., K = 3



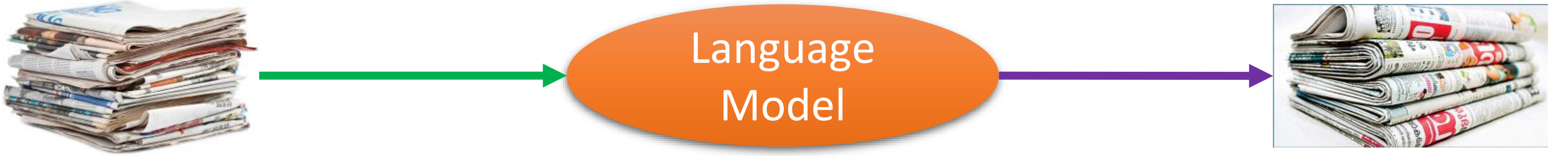
A small number of events occur with high frequency
A large number of events occur with low frequency

https://en.wikipedia.org/wiki/Zipf%27s_law

Let’s re-look at testing

Training set (Straits Times, Jan – Sept)

Test set (Straits Times, Oct – Dec)



denied allegations: 5
denied speculation: 2
denied rumours: 1
denied report: 1

“denied offer……
“denied loan …..

- Both words “offer” and “loan” are common words (there is no unknown words)
- But, the model does not see “denied offer” or “denied loan” in training
  - *P(offer|denied) = 0?*   *P(loan|denied) = 0?*

The training data is **never large enough** to cover *all* possible word combinations!

Smoothing: avoid assigning zero probabilities to unseen events

- There are many smoothing techniques available
  - **Laplace (add-one) smoothing**
  - Add-k smoothing
  - Stupid backoff
  - Kneser-Ney smoothing

- The simplest way to do smoothing: **Laplace Smoothing**
  - Assuming every possible n-gram appears once which we do not explicitly observe.
  - **Add one to all the n-gram counts, before we normalize them into probabilities**.

  - Laplace smoothing does not perform well enough to be used smoothing in modern n-gram models
    - But it usefully introduces many of the concepts in other smoothing algorithms,
    - Is a practical smoothing algorithm for other tasks like text classification

Laplace Smoothing Example

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

Raw counting

	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

$$P(want|I) = \frac{C(I\ want)}{C(I)} = \frac{827}{2533}$$

Laplace smoothing (add-one)

	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

$$P(want|I) = \frac{C(I\ want) + 1}{C(I) + ?} = \frac{828}{2533 + ?}$$

Laplace Smoothing Example

	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

(I I)
(I want)
(I to)
(I Chinese)
(I food)
(I lunch)
(I spend)
(I …)
….

(want I)
(to I)
(Chinese I)
(food I)
(lunch I)
(spend I)
(… I)
….

(I, every word in vocabulary)

- What are added to the training text?
  - For each word, we have added |V| number of additional appearance.
  - |V| is the size of vocabulary.

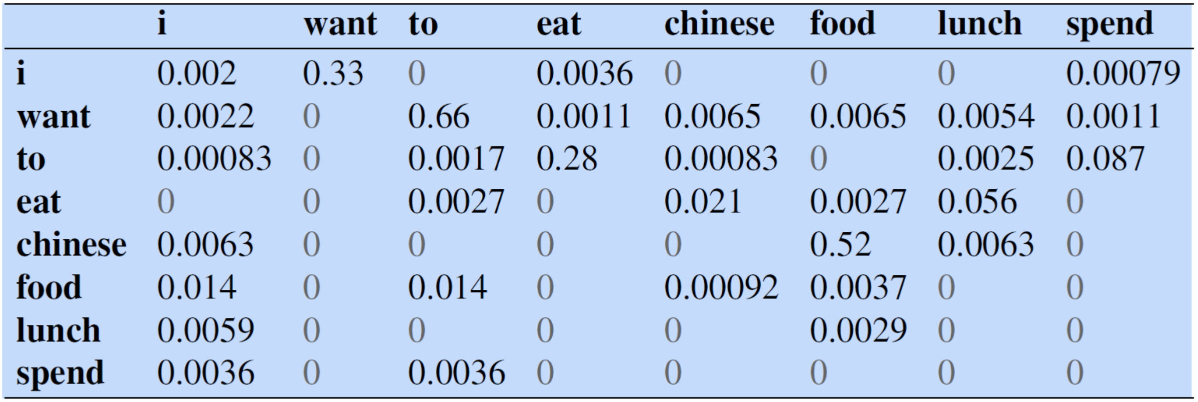
i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

$$P_{laplace}(want|I) = \frac{C(I\ want) + 1}{C(I) + V} = \frac{828}{2533 + 1446}$$

In this example dataset: Number of unique words (vocabulary): 1446

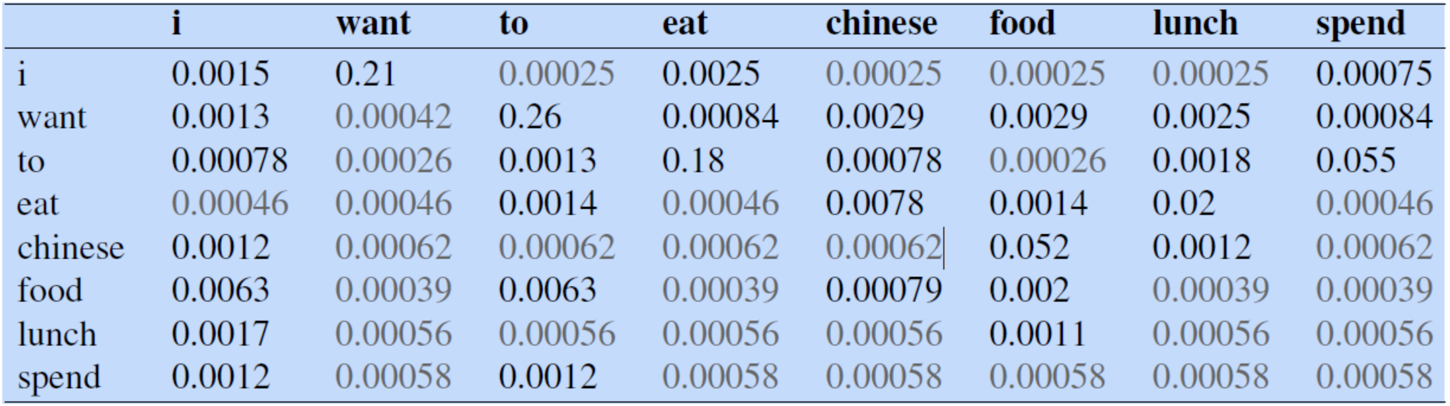
Laplace Smoothing

- Before smoothing
  - $P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$
- With Laplace smoothing
  - $P_{laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$



	i	want	to	eat	chinese	food	lunch	spend
i	0.002	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

$P(to|want)$ changed from 0.66 to 0.26!
→ too much probability mass is moved to all the zeros.



	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-k Smoothing

- Move a bit less of the probability mass from the seen to the unseen events.
- Instead of adding 1 to each count, we add a fractional count k $(0 < k < 1)$

$$P_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

- The value of k can be $0.1, 0.05, 0.5$ or other values, determined on a development set.

- Add-k is useful for some tasks like text classification, but in general does not do well for language modeling

Backoff and Interpolation

- Backoff: use less context
  - We use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram.
  - We only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram.

- Interpolation
  - We always mix the probability estimates from all the n-gram estimators, weighting and combining the trigram, bigram, and unigram counts.
  - $\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2}w_{n-1})$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$

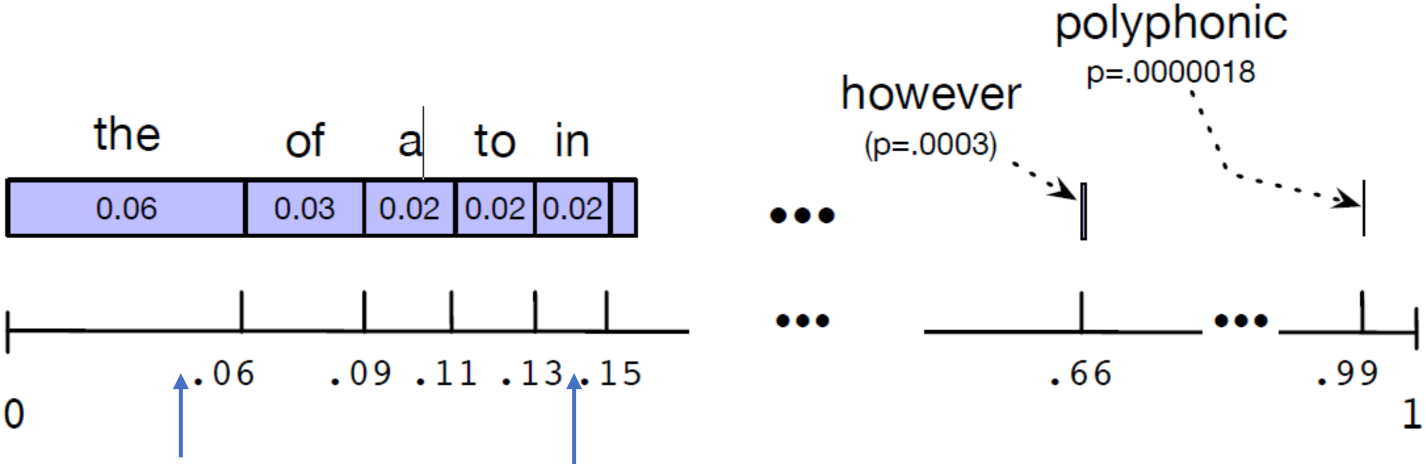
Language Generation

- Sampling sentences from a language model
  - Sampling from a distribution: choose random points according to their likelihood.
  - Sampling from a language model, which represents a distribution over sentences, means to generate some sentences: choose each sentence according to its likelihood

- Visualize how sentence generation works for the unigram case.
  - A unigram language model is a collection of unigram probabilities, e.g., $P(the), P(of), P(a)$ … .
  - We place these unigram probabilities on a line, ordered by their probabilities
  - Then we generate a random value between 0 and 1, find the point on the probability line, and print the word whose interval includes this random value

Language Generation (for unigram case)

- Visualize how sentence generation works for **the unigram case**.
  - A unigram language model is a collection of unigram probabilities, e.g., $P(the), P(of), P(a) \ldots.$
  - We place these unigram probabilities on a line, ordered by their probabilities
  - Then we generate a random value between 0 and 1, find the point on the probability line, and print the word whose interval includes this random value



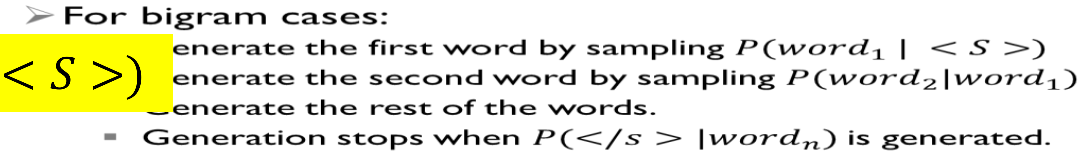
Random value: 0.05 → word: the

0.14 → word: in

Language Generation (for bigram case)

- For bigram cases:
  - Generate the first word by sampling $P(word_1 \mid < S >)$
  - Generate the second word by sampling $P(word_2|word_1)$
  - Generate the rest of the words.
  - Generation stops when $P(</s > |word_n)$ is generated.

$P(word_1| < S >)$



- For bigram cases:
  - enerate the first word by sampling $P(word_1 \mid < S >)$
  - enerate the second word by sampling $P(word_2|word_1)$
  - enerate the rest of the words.
  - Generation stops when $P(</s > |word_n)$ is generated.

Random value: 0.05 → word: the

0.14 → word: in

N-gram Language Model

- Word prediction
  - Probability of a sequence of words $P(w_1 w_2 \ldots . w_n)$, or probability of a word given some history $P(w|h)$
- N-grams
  - Counting and basic concepts
- N-gram Language Model
  - Modeling unknown words
  - Smoothing to avoid assigning zero probabilities to unseen sequences
  - Evaluation
- Reference: https://web.stanford.edu/~jurafsky/slp3/
  - **Chapter 3**, N-gram Language Models

What can we do?

- Given a collection of documents, we are able to train a language model

- Given a language model, we are able to compute the probability of sentences

- Given a language model, we can also generate sentences