

# Probabilistic Language Models

Ramaseshan Ramachandran

A Brief Introduction to probability  
Probabilistic Language Model -  
Definition  
Chain Rule  
Markov Assumption

Ramaseshan

Target and Context words  
Language Modeling using Unigrams  
Generative Model  
Maximum Likelihood Estimate  
Bigram Language Model  
Bigram Language Model - Example

## INTRODUCTION

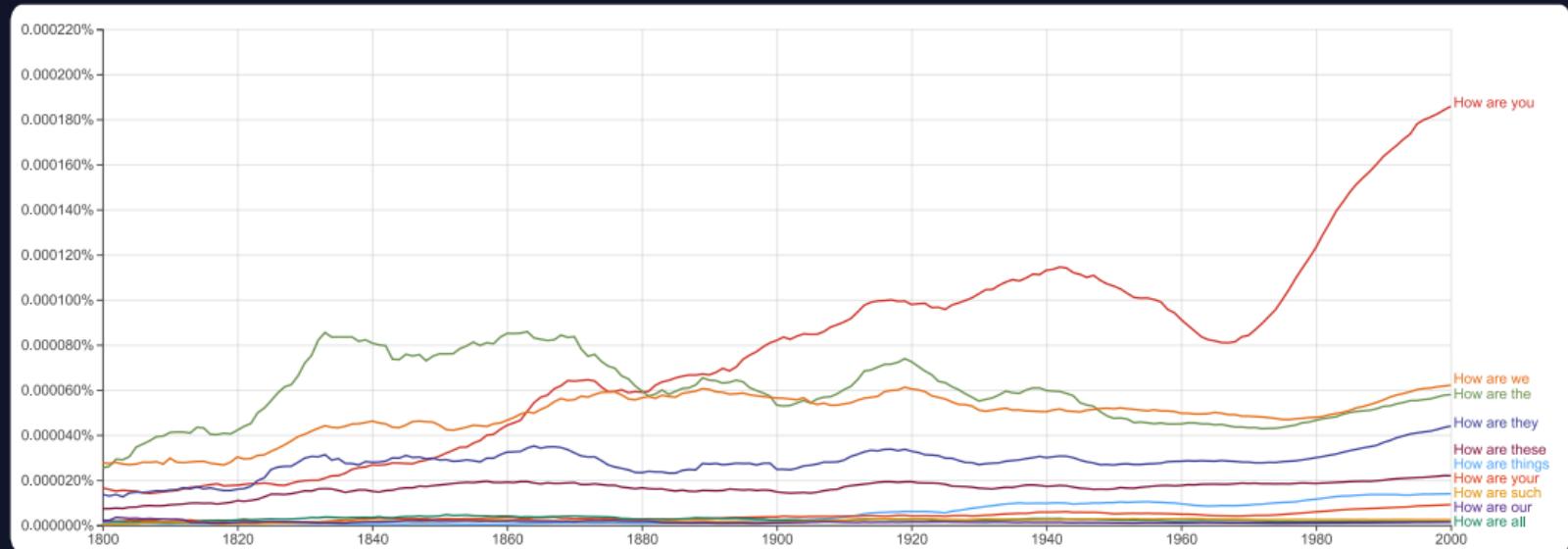
---

How are \_\_\_\_? Can you guess the missing word?

Ramaseshan

# INTRODUCTION

How are \_\_\_\_? Can you guess the missing word?



Source: Google NGram Viewer

## INTRODUCTION

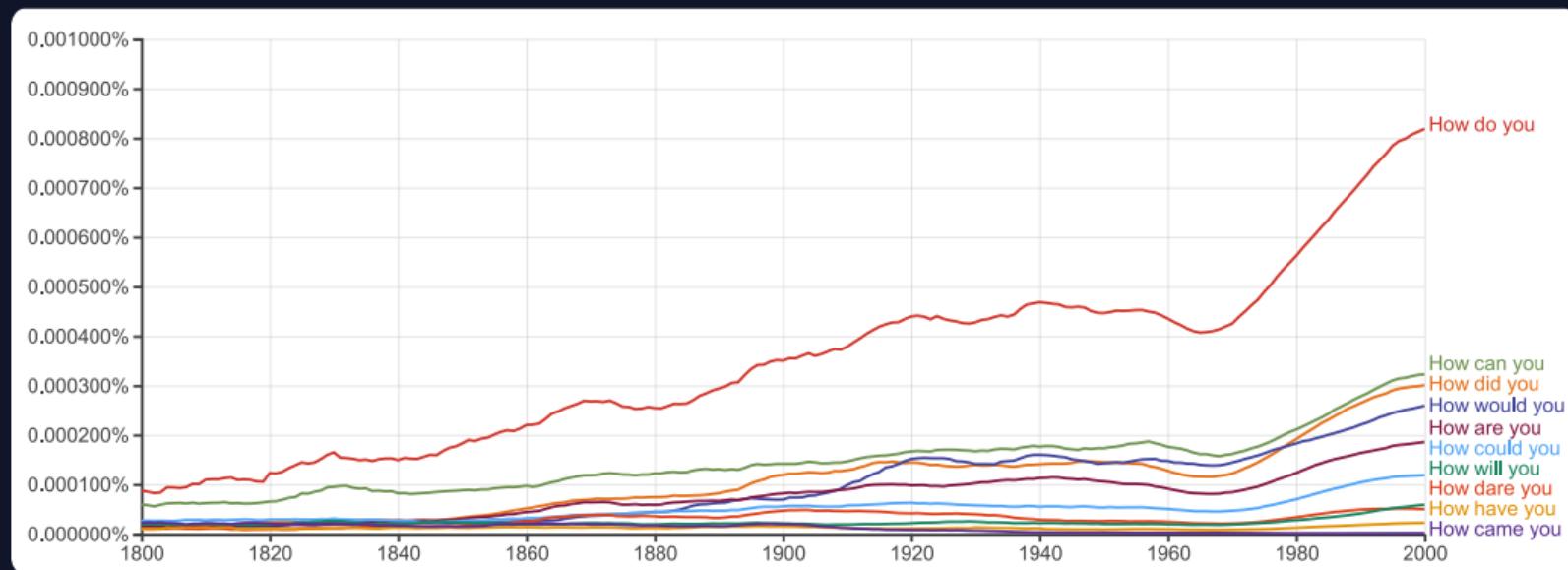
---

How \_\_\_ you? Can you guess the missing word?

Ramaseshan

# INTRODUCTION

How \_\_\_ you? Can you guess the missing word?



Source: Google NGram Viewer

## INTRODUCTION

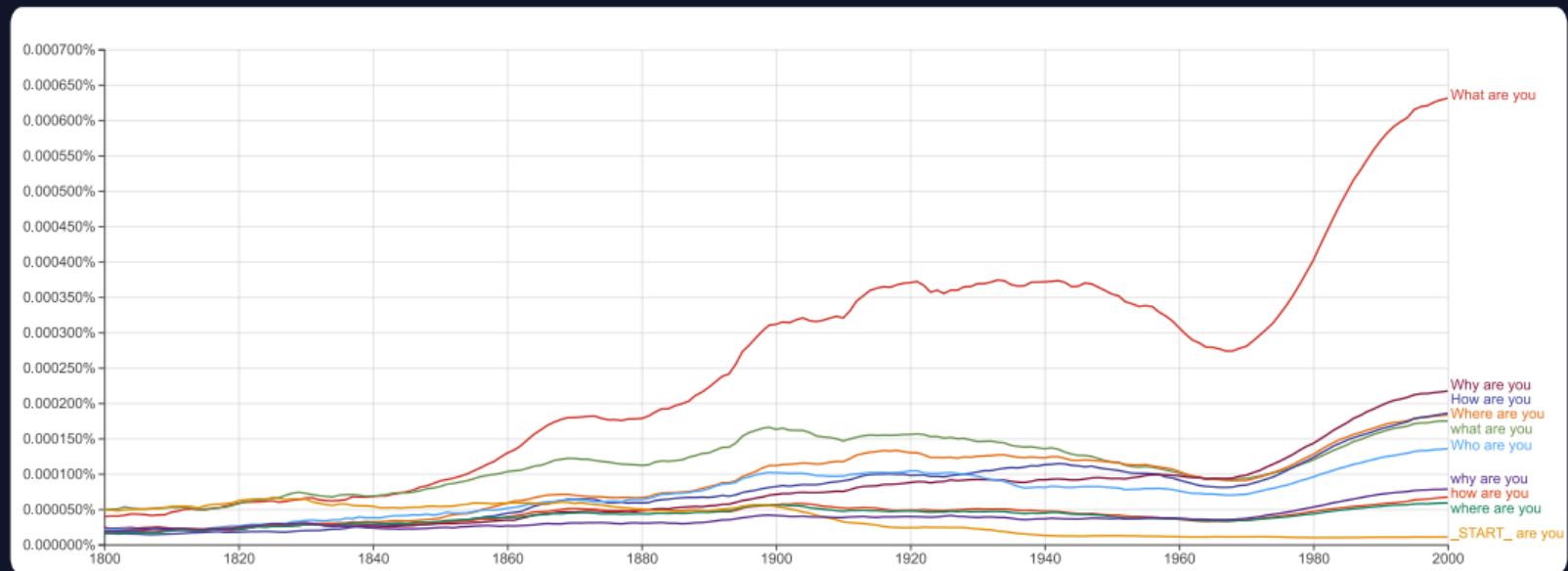
---

\_\_\_ are you?

Ramaseshan

# INTRODUCTION

\_\_\_ are you?



Source: Google NGram Viewer

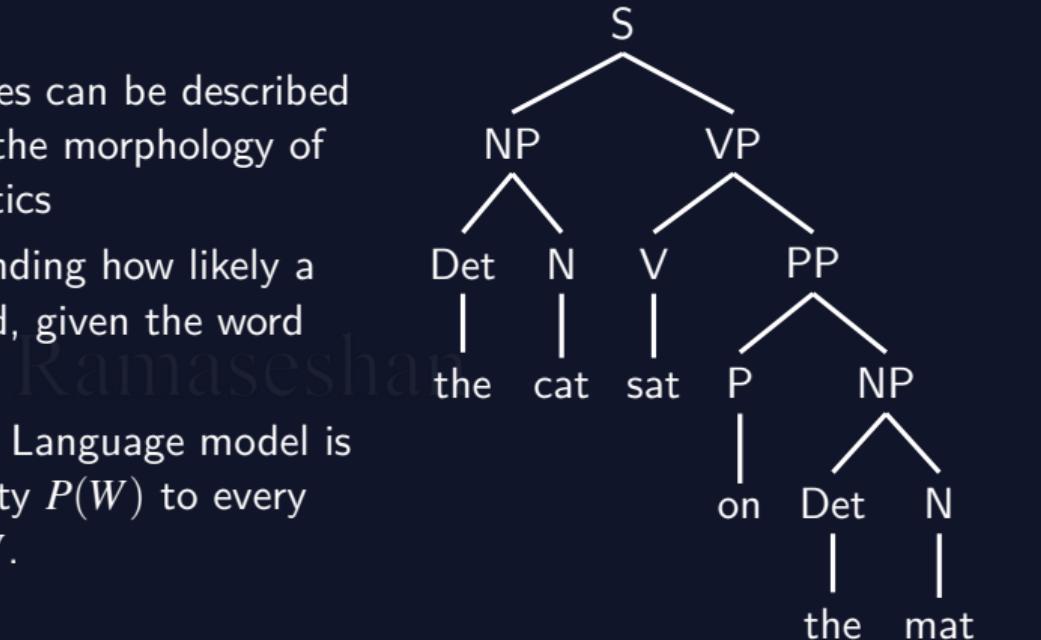
How do humans predict the next word?

- ▶ Domain knowledge
- ▶ Syntactic knowledge
- ▶ Lexical knowledge
- ▶ Knowledge about the sentence structure
- ▶ Some words are hard to find. Why?
- ▶ Natural language is not deterministic in general
- ▶ Some sentences are familiar or had been heard/seen/used several times
- ▶ They are more likely to happen than others, hence we could guess

# THE LANGUAGE MODEL

---

- ▶ Natural language sentences can be described by parse trees which use the morphology of words, syntax and semantics
- ▶ Probabilistic thinking - finding how likely a sentence occurs or formed, given the word sequence.
- ▶ In probabilistic world, the Language model is used to assign a probability  $P(W)$  to every possible word sequence  $W$ .



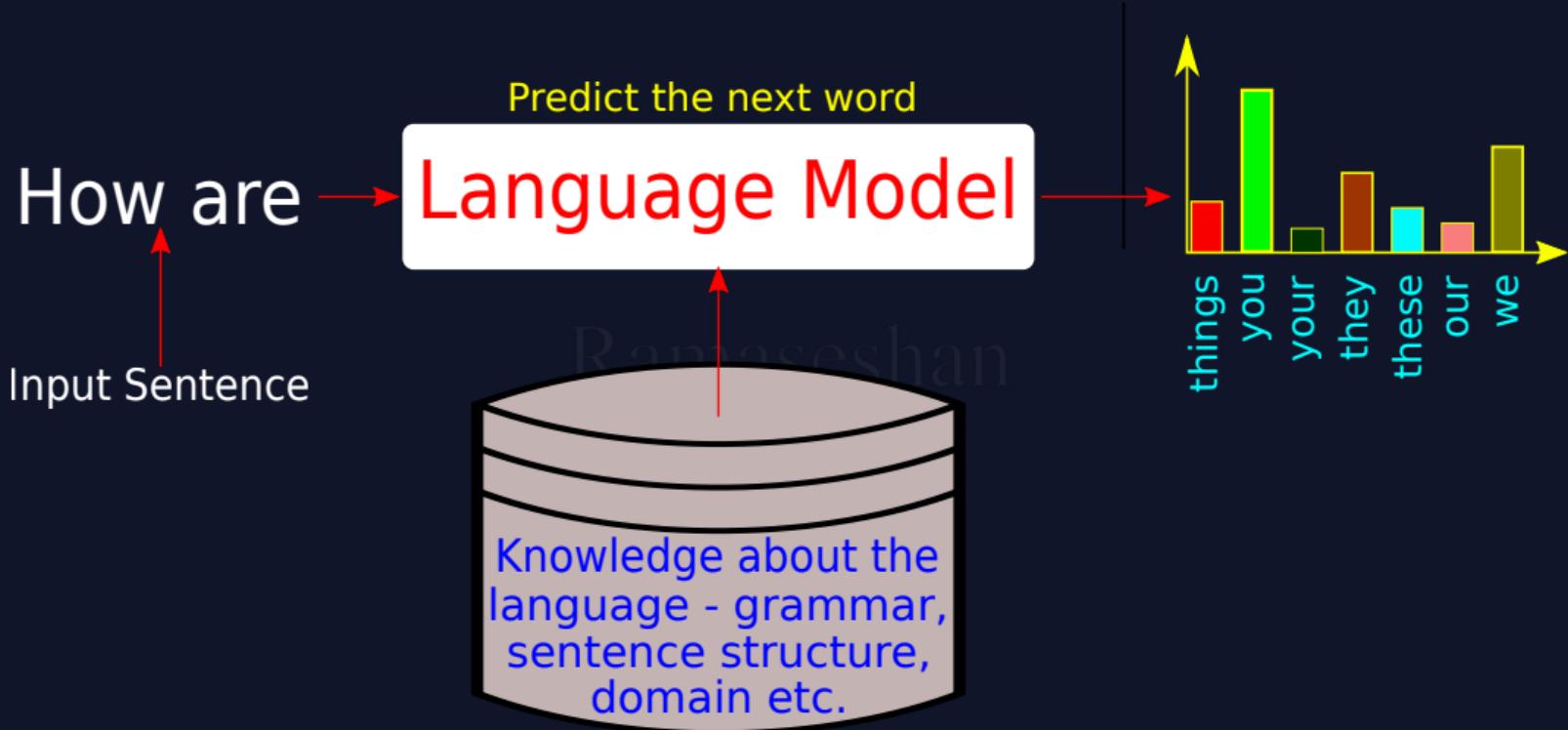
The current research in Language models focuses more on building the model from the huge corpus of text

# APPLICATIONS

---

Application	Sample Sentences
Speech Recognition	Did you hear <i>Recognize speech</i> or Wreck a nice beach?
Context sensitive Spelling	One upon a <i>tie</i> , <i>Their</i> lived a king
Machine translation	artwork is good → l'oeuvre est bonne
Sentence Completion	Complete a sentence as the previous word is given - GMail

# A SIMPLE LANGUAGE MODEL IMPLEMENTATION



## WHY PROBABILISTIC MODEL

---

- ▶ Speech recognition systems cannot depend on the processed speech signals. It may require the help of a language model and context recognizer to convert a speech to correct text format.
- ▶ As there are multiple combinations for a word to be in the next slot in a sentence, it is important for language modeling to be probabilistic in nature - judgment about the fluency of a sequence of words returns the probability of the sequence
- ▶ The probability of the next word in a sequence is real number  $[0, 1]$
- ▶ The combination of words with high-probability in a sentence are more likely to occur than low-probability ones
- ▶ A probabilistic model continuously estimates the rank of the words in a sequence or phrase or sentence in terms of frequency of occurrence

## FORMAL DEFINITION

---

Let  $\mathcal{V}$  be the vocabulary, a finite set of symbols or words. Let us use  $\triangleleft$  and  $\triangleright$  as the start and stop symbols and let them be the part of  $\mathcal{V}$ . Let  $|\mathcal{V}|$  denote the size of  $\mathcal{V}$ .

Let  $W$  be infinite sequences of words from the collection of  $\mathcal{V}$ . Every sequence in  $W$  starts with  $\triangleleft$  and ends with  $\triangleright$ . Then a language model is a probability distribution of a random variable  $X$  which takes values from  $W$ . Or  $p: W \rightarrow \mathbb{R}$  such that

$$\forall x \in W, p(x) \geq 0 \text{ and} \tag{1}$$

$$\sum_{x \in W} p(X = x) = 1 \tag{2}$$

**Goal:** Compute the probability of a sequence of words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) \quad (3)$$

**Task:** To predict the next word using probability. Given the context, find the next word using

$$P(w_n | w_1, w_2, w_3, \dots, w_{n-1}) \quad (4)$$

A model which computes the probability for (3) or predicting the next word (4) or complete the partial sentence is called as Probabilistic Language Model.

The goal is to learn the joint probability function of sequences of words in a language. The probability of  $P(\text{The cat roars})$  is less likely to happen than  $P(\text{The cat meows})$

The chain rule states that the probability of a sequence of events occurring is the product of the probabilities of each event occurring. In the context of NLP, The sentence "I am a large language model" occurring is the product of the probabilities of the words "I", "am", "a", "large", "language", and "model" occurring in the corpus.

- ▶ Chain rule models joint probability of multiple events in NLP.
- ▶ For word sequence  $(w_1, w_2, \dots, w_n)$ :

$$P(w_1, w_2, \dots, w_n) = P(w_1) \cdot P(w_2|w_1) \cdot \dots \cdot P(w_n|w_1, w_2, \dots, w_{n-1})$$

- ▶ Breaks down into product of conditional probabilities.
- ▶ Essential for language models (n-grams, Markov models, RNNs, transformers).
- ▶ Enables language sequence modeling and likelihood estimation.

## MARKOV ASSUMPTION

---

**Markov Assumption:** The future behavior of a dynamic system depends on its recent history and not on the entire history

The product of the conditional probabilities can be written approximately for a bigram as

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-1}) \quad (5)$$

Equation (5) can be generalized for an *n*-gram as

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-K+1}^{k-1}) \quad (6)$$

Now, the joint probability of a sequence can be re-written as

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) = P(w_1^n) \quad (7)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \dots, w_1) \quad (8)$$

$$= \prod_{k=1}^n P(w_k|w_1^{k-1}) \quad (9)$$

$$\approx \prod_{k=1}^n P(w_k|w_{k-K+1}^{k-1}) \quad (10)$$

## TARGET AND CONTEXT WORDS

---

Next word in the sentence depends on its immediate past words, known as context words

$$P(w_{k+1} | \underbrace{w_{i-k}, w_{i-k+1}, \dots, w_k}_{\text{Context words}})$$

n-grams

unigram -  $P(w_{k+1})$

bigram -  $P(w_{k+1} | w_k)$

trigram -  $P(w_{k+1} | w_{k-1}, w_k)$

4-gram -  $P(w_{k+1} | w_{k-2}, w_{k-1}, w_k)$

Ramaseshan

- ▶ All words are generated independent of its history  $W, W_2, W_3, \dots, W_n$  and none of them depend on the other
- ▶ Not a good model for language generation
- ▶ It will have  $|V|$  parameters
- ▶  $\theta_i = p(w_i) = \frac{c_{w_i}}{N}$ , where  $c_{w_i}$  if the count of the word  $w_i$  and  $N$  is the total number of words in the vocabulary
- ▶ It may not be able to pick up regularities present in the corpus
- ▶ It is more likely to generate **the the the the** as a sentence than a grammatically valid sentence

- ▶ Generates a document containing  $N$  words using n-gram
- ▶ A good model assigns higher probability to the word that actually occurs

$$P(\mathbf{W}) = P(N) \prod_{i=1}^N P(W_i) \quad (11)$$

Ramaseshan

- ▶ The location of the word in the document is not important
- ▶  $P(N)$  is the distribution over  $N$  and is same for all documents. Hence it may be ignored
- ▶  $W_i$ , to be estimated in this model is  $P(W_i)$  and it must satisfy  $\sum_{i=1}^N P(w_i) = 1$

## MAXIMUM LIKELIHOOD ESTIMATE

---

- ▶ One of the methods to find the unknown parameter(s) is the use of Maximum Likelihood Estimate
- ▶ Estimate the parameter value for which the observed data has the highest probability
- ▶ Training data may not have all the words in the vocabulary
- ▶ If a sentence with an unknown word is presented, then the MLE is zero.
- ▶ Add a smoothing parameter to the equation without affecting the overall probability requirements

$$P(\mathbf{W}) = \frac{C_{w_i} + \alpha}{C_W + \alpha|V|} \quad (12)$$

If  $\alpha = 1$ , then it is called as Laplace smoothing (13)

$$P(\mathbf{W}) = \frac{C_{w_i} + 1}{C_W + |V|} \quad (14)$$

- ▶ This model generates a sequence one word at a time, starting with the first word and then generating each succeeding word conditioned on the previous one or its predecessor
- ▶ A bigram language model or the Markov model (first order) is defined as follows:

Ramaseshan

$$P(\mathbf{W}) = \prod_{i=1}^{n+1} P(w_i | w_{i-1}) \quad (15)$$

where  $\mathbf{W} = w_1, w_2, w_3, \dots, w_n$

- ▶ Estimate the parameter  $P(w_i|w_{i-1})$  for all bigrams
- ▶ The parameter estimation does not depend on the location of the word
- ▶ If we consider the sentence as a sequence in time, they are time-invariant MLE picks up the word that is  $\frac{n_{w,w'}}{n_{w,o}}$  where  $n_{w,w'}$  is the number of times the words  $w, w'$  occur together and  $n_{w,o}$  is the number of times the word  $w$  appears in the bigram sequence with any other word
- ▶ The number of parameters to be estimated =  $|V| \times (|V| + 1)$

## PROBABILISTIC LANGUAGE MODEL - EXAMPLE

Peter Piper picked a peck of pickled peppers  
A peck of pickled peppers Peter Piper picked  
If Peter Piper picked a peck of pickled peppers  
Where's the peck of pickled peppers Peter Piper picked?  
—

The joint probability of a sentence formed with  $n$  words can be expressed as a product conditional probabilities - we use immediate context and not the entire history

$$P(w_1 | \langle \rangle) \times P(w_2 | w_1) \times \dots \times P(\langle E \rangle | w_n)$$

$$\text{and } P(w_{i+1} | w_i) = \frac{C(w_i, w_{i+1})}{C(w_i)}$$
  
—

What is the probability of these sentences?

$P(\text{Peter Piper picked})$

$P(\text{Peter Piper picked peppers})$

Bigram	Frequency
$\langle \rangle \text{peter}$	1
peter peter	4
peter picked	4
picked a	2
a peck	2
peck of	4
pickled peppers	4
peppers $\rangle$	1
$\langle \rangle \text{a}$	1
a peck	1
peck of	1
of pickled	4
peppers peter	2
...	..
$\langle \dots$	1

## BUILDING A BIGRAM MODEL - CODE

---

```
1 #compute the bigram model
2 def build_bigram_model():
3     bigram_model = collections.defaultdict(
4         lambda: collections.defaultdict(lambda: 0))
5     for sentence in kinematics_corpus.sents():
6         sentence = [word.lower() for word in sentence
7                     if word.isalpha()] # get alpha only
8     #Collect all bigrams counts for (w1,w2)
9     for w1, w2 in bigrams(sentence):
10        bigram_model[w1][w2] += 1
11    #compute the probability for the bigram containing w1
12    for w1 in bigram_model:
13        #total count of bigrams conaining w1
14        total_count = float(sum(bigram_model[w1].values()))
15        #distribute the probability mass for all bigrams starting with w1
16        for w2 in bigram_model[w1]:
17            bigram_model[w1][w2] /= total_count
18
19 return bigram_model
```

## BUILDING A BIGRAM MODEL - CODE

```
def predict_next_word(first_word):
    #build the model
    model = build_bigram_model()
    #get the next for the bigram starting with 'word'
    second_word = model[first_word]
    #get the top 10 words whose first word is 'first_word'
    top10words = Counter(second_word).most_common(10)

    predicted_words = list(zip(*top10words))[0]
    probability_score = list(zip(*top10words))[1]
    x_pos = np.arange(len(predicted_words))

    plt.bar(x_pos, probability_score, align='center')
    plt.xticks(x_pos, predicted_words)
    plt.ylabel('Probability Score')
    plt.xlabel('Predicted Words')
    plt.title('Predicted words for ' + first_word)
    plt.show()

predict_next_word('how')
```

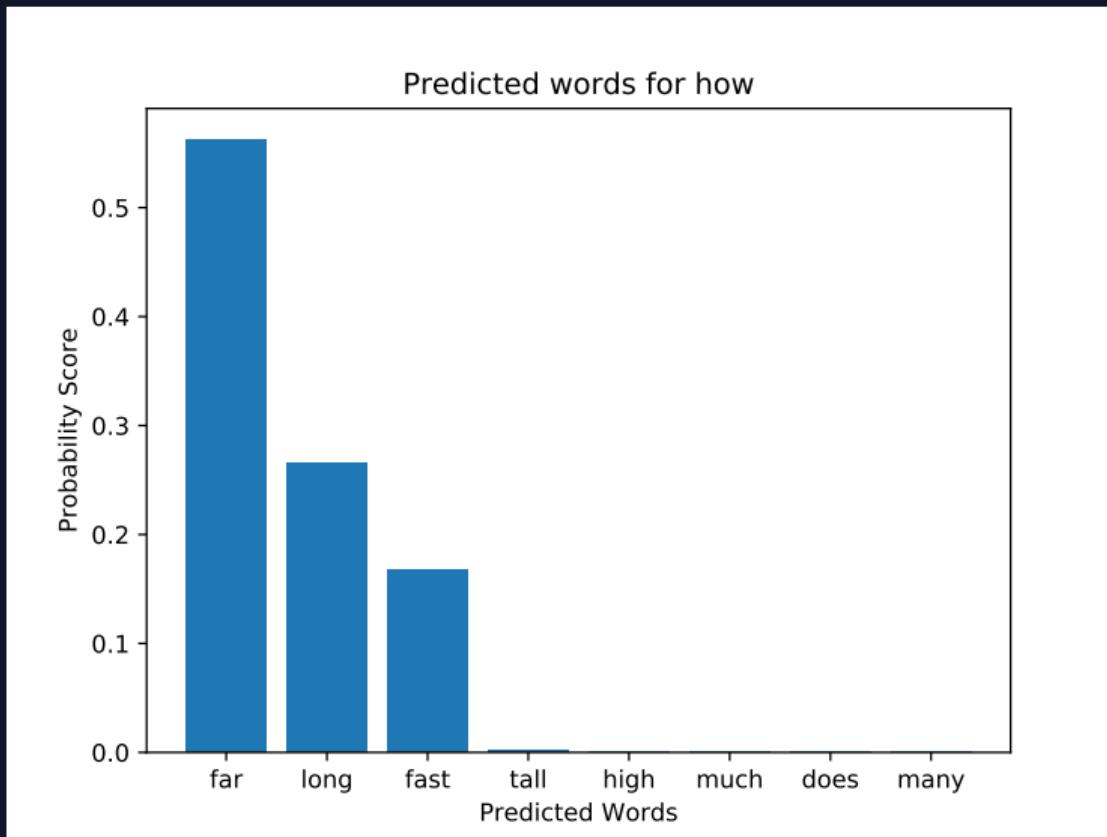
# MODEL PARAMETERS - BIGRAM EXAMPLE

The screenshot shows the PyCharm IDE interface with the following components:

- Editor:** Displays the `BigramLM.py` file containing Python code for building a bigram language model. The code includes functions for building the model, predicting the next word, and plotting predicted words against probability scores.
- Variable Explorer:** Shows the state of variables in memory. It lists `corpusdir` (str), `first_word` (str), and `model` (defaultdict). The `model` variable is expanded to show its contents.
- Model - Dictionary (926 elements) Window:** A detailed view of the `model` defaultdict. It shows 926 entries, with `how` selected. The inner dictionary `how - Dictionary (8 elements)` is also shown, listing words like `does`, `far`, `fast`, etc., along with their float values.
- IPython console:** Shows the history of commands entered in the debugger. It includes the creation of the `model` defaultdict, the assignment of `model['accident']` to `defaultdict(<function build_bigram_model.<locals>.<lambda>.<locals>.<lambda> at 0x7f86d2280c80, {'note': 1.0})`, and three consecutive `ipdb>` prompts.

```
File Edit Search Source Run Debug Console Projects Tools View Help
File Edit Search Source Run Debug Console Projects Tools View Help
Editor - /home/ramaseshan/PycharmProjects/Class2019/BigramLM.py
LSI_Gensim.py X utils.py X UnigramLM.py X BigramLM.py X
20     if word.isalpha()]: # get alpha only
21         #Collect all bigrams counts for (w1,w2)
22         for w1, w2 in bigrams(sentence):
23             bigram_model[w1][w2] += 1
24         #compute the probability for the bigram starting with w1
25         for w1 in bigram_model:
26             #total count of bigrams starting with w1
27             total_count = float(sum(bigram_model[w1]))
28             #distribute the probability mass for w1
29             for w2 in bigram_model[w1]:
30                 bigram_model[w1][w2] /= total_count
31
32     return bigram_model
33
34 def predict_next_word(first_word):
35     #buikd the model
36     model = build_bigram_model()
37     #get the next for the bigram starting with first_word
38     second_word = model[first_word]
39     #get the top 10 words whose first word is 'first_word'
40     top10words = Counter(second_word).most_common(10)
41
42     predicted_words = list(zip(*top10words))[0]
43     probability_score = list(zip(*top10words))[1]
44     x_pos = np.arange(len(predicted_words))
45
46     # calculate slope and intercept for the linear trend line
47     slope, intercept = np.polyfit(x_pos, probability_score, 1)
48
49     plt.bar(x_pos, probability_score, align='center')
50     plt.xticks(x_pos, predicted_words)
51     plt.ylabel('Probability Score')
52     plt.xlabel('Predicted Words')
53     plt.title('Predicted words for ' + first_word)
54     plt.show()
55
56
Name          Type      Size   Value
corpusdir    str       1      /home/ramaseshan/Dropbox/NLPClass/2019/Co...
first_word   str       1      how
model        defaultdict 926    defaultdict object of collections module
model - Dictionary (926 elements)
Key          Type      Size   Value
hotel        defaultdict 2      defaultdict object...
hour         float     1      5.139921410301656e-19
hours        float     1      0.5628668144533753
house        float     1      0.16799166845157743
houston      float     1      0.0009765637999710939
hovering    float     1      0.26565957263477835
how          float     1      5.019455100613326e-22
hr           float     1      0.0005502887070202716
human        float     1      1e-100
i             float     1      1e-100
icepe        float     1      1e-100
i.e.         float     1      1e-100
Save and Close Close
ipdb>
ipdb>
ipdb> model['accident']
defaultdict(<function build_bigram_model.<locals>.<lambda>.<locals>.<lambda> at 0x7f86d2280c80, {'note': 1.0})
ipdb>
ipdb>
ipdb>
Permissions: RW End-of-lines: LF Encoding: ASCII Line: 38 Column: 1 Memory: 38%
24 / 46
```

## BIGRAM MODEL - NEXT WORD PREDICTION



## MODEL PARAMETERS - TRIGRAM EXAMPLE

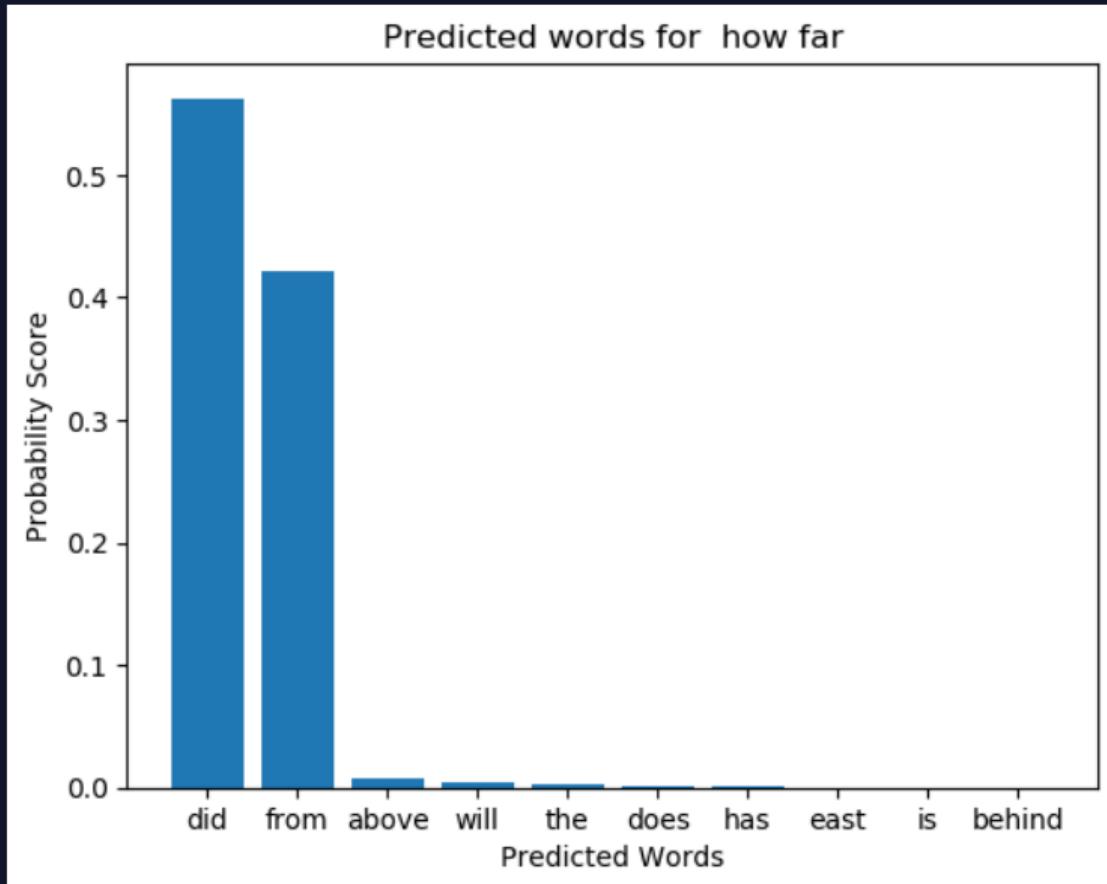
The screenshot shows the PyCharm IDE interface with the following components:

- File Bar:** File, Edit, Search, Source, Run, Debug, Console, Projects, Tools, View, Help.
- Toolbar:** Standard file operations like Open, Save, Run, Stop, etc.
- Editor:** The main code editor window containing Python code for generating trigrams and predicting the next word. The code uses the `collections` module and `Counter` class.
- Variable Explorer:** A table showing the state of variables. It includes columns for Name, Type, Size, and Value. Key entries include `corpusdir` (str, 1, value: /home/ramaseshan/Dropbox/NLPClass/2019/Corpus/), `model` (defaultdict, 3668, value: defaultdict object of collections module), `w1` (str, 1, value: how), and `w2` (str, 1, value: far).
- IPython Console:** An open terminal window showing the history of commands entered. The command `predict\_next\_word('how', 'far')` was run, and the output is displayed in two tables:

  - model - Dictionary:** Shows the trigram model as a dictionary where keys are tuples of words and values are floats representing probability scores. Examples include ('how', 'far'): 0.0078125, ('far', 'will'): 6.357828776041666e-07, and ('will', 'he'): 1.271565755208333e-06.
  - (how, 'far') - Dictionary (14 elements):** Shows the top 14 most probable words following 'how' and 'far' with their respective probability scores.

- Bottom Status Bar:** IPython console and History log tabs.

## TRIGRAM MODEL - NEXT WORD PREDICTION



- ▶ In a closed vocabulary language model, there is no unknown words or ***out of vocabulary words (OOV)***
- ▶ In an open vocabulary system, you will find new words that are not present in the trained model
- ▶ Pick words below certain frequency and replace them as OOV.
- ▶ Treat every OOV as a regular word
- ▶ During testing, the new words would be treated as OOV and the corresponding frequency will be used for computation
- ▶ This eliminates zero probability for sentences containing OOV

Recap of

# Probabilistic Language Model

Ramaseshan

The chain rule allows us to compute probabilities using *N-gram* models. It states that the joint probability of a sequence can be expressed as the product of conditional probabilities. The chain rule is defined as:

$$P(w_1, w_2, \dots, w_N) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \dots P(w_N|w_1, w_2, \dots, w_{N-1}) \quad (16)$$

### Example

Consider a bigram model. To calculate the probability of a sentence, say "I love cats," we can use the chain rule:  $P(\text{"I love cats"}) = P(\text{"I"}) \cdot P(\text{"love"}|\text{"I"}) \cdot P(\text{"cats"}|\text{"love"})$ .

Given a training corpus of sentences, we can calculate the MLE probabilities.

Let  $w_i$  represent the  $i$ -th word in a sentence

$w_1^{i-1}$  denote the context words before  $w_i$

$$P(w_i|w_1^{i-1}) = \frac{\text{count}(w_1^{i-1}, w_i)}{\text{count}(w_1^{i-1})} \quad (17)$$

The MLE probability  $P(w_i|w_1^{i-1})$  is estimated by counting the occurrences of  $w_i$  with the specific context and normalizing by the total count of that context: Here,  $\text{count}(w_1^{i-1}, w_i)$  represents the number of times the context  $w_1^{i-1}$  is followed by the word  $w_i$ , and  $\text{count}(w_1^{i-1})$  is the count of the context  $w_1^{i-1}$ .

Let's define the Bigram Language Model equation:

$$P(W) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \dots \cdot P(w_n|w_{n-1}) \quad (18)$$

where

- ▶  $P(W)$ : Probability of observing the entire sequence  $W$ .
- ▶  $P(w_i)$ : Probability of the  $i$ th word in the sequence.
- ▶  $P(w_i|w_{i-1})$ : Conditional probability of the  $i$ th word given the  $(i-1)$ th word.

## BIGRAM LANGUAGE MODEL EXAMPLE

---

Now, let's consider a small corpus of sentences:

1. "I love cats."
2. "Cats love playing."
3. "Dogs chase cats."
4. "Many love cats."

We will build a bigram language model to calculate the probabilities of these sentences.  
The probabilities for each sentence can be calculated as follows:

$$P(\text{"I love cats."}) = P(\text{I}) \cdot P(\text{love} \mid \text{I}) \cdot P(\text{cats} \mid \text{love}) \cdot P(\text{.} \mid \text{cats}) \quad (19)$$

$$P(\text{"Cats love playing."}) = P(\text{cats}) \cdot P(\text{love} \mid \text{cats}) \cdot P(\text{playing} \mid \text{love}) \cdot P(\text{.} \mid \text{playing}) \quad (20)$$

$$P(\text{"Dogs chase cats."}) = P(\text{dogs}) \cdot P(\text{chase} \mid \text{dogs}) \cdot P(\text{cats} \mid \text{chase}) \cdot P(\text{.} \mid \text{cats}) \quad (21)$$

## COMPUTING PROBABILITIES

---

To calculate the probabilities, we need to count the occurrences of each word and each bigram in the corpus. For example, for the sentence "I love cats.":

►  $\text{Count}(I) = 1 ; \text{Count}(\text{love} | I) = 1 ; \text{Count}(\text{cats} | \text{love}) = 2 ; \text{Count}(. | \text{cats}) = 1$

Using Maximum Likelihood Estimation (MLE), we can calculate the probabilities as follows:

$$P(I) = \frac{\text{Count}(I)}{\text{Total number of words in the corpus}} \quad (22)$$

$$P(\text{love} | I) = \frac{\text{Count}(\text{love} | I)}{\text{Count}(I)} \quad (23)$$

$$P(\text{cats} | \text{love}) = \frac{\text{Count}(\text{cats} | \text{love})}{\text{Count}(\text{love})} \quad (24)$$

$$P(. | \text{cats}) = \frac{\text{Count}(. | \text{cats})}{\text{Count}(\text{cats})} \quad (25)$$

Finally, we can plug these values into the bigram language model equation to estimate the probability of each sentence.

- ▶ A measure to evaluate a language model
- ▶ Quantifies how well a language model predicts the next word in a sequence
- ▶ Lower the perplexity implies a better language model

The perplexity of the language model on the test set is a measure of how well the language model generalizes to unseen text.

## PERPLEXITY OF A BIGRAM MODEL

---

The perplexity of a bigram model is calculated using the following equation:

$$\begin{aligned} \text{perplexity} &= \exp \left( -\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_{i-1}) \right) \\ &= \exp \left( -\frac{1}{N} \sum_{i=1}^N \log \left( \frac{\text{count}(w_i, w_{i-1})}{\text{count}(w_{i-1})} \right) \right) \end{aligned} \quad (26)$$

where

*perplexity* is the measure of the language model

*N* is the number of words in the test set

$w_i$  is the *i*th word in the test set

$w_{i-1}$  is the  $(i-1)$ th word in the test set

$\text{count}(w_i, w_{i-1})$  is the number of times the bigram  $(w_i, w_{i-1})$  appears in the training corpus

$\text{count}(w_{i-1})$  is the number of times the word  $w_{i-1}$  appears in the training corpus

The perplexity equation can be used to evaluate the performance of a bigram language model on a test set.

## PERPLEXITY OF A TRIGRAM MODEL

---

The perplexity of a trigram model is calculated using the following equation:

$$\text{perplexity} = \exp \left( -\frac{1}{N} \sum_{i=1}^N \log \left( \frac{\text{count}(w_i, w_{i-1}, w_{i-2})}{\text{count}(w_{i-1}, w_{i-2})} \right) \right) \quad (27)$$

where:

*perplexity* is the perplexity of the language model

*N* is the number of words in the test set

$w_i$  is the *i*th word in the test set

$w_{i-1}$  is the  $(i-1)$ th word in the test set

$w_{i-2}$  is the  $(i-2)$ th word in the test set

$\text{count}(w_i, w_{i-1}, w_{i-2})$  is the number of times the trigram  $(w_i, w_{i-1}, w_{i-2})$  appears in the training corpus

$\text{count}(w_{i-1}, w_{i-2})$  is the number of times the bigram  $(w_{i-1}, w_{i-2})$  appears in the training corpus

The only difference is that the trigram language model uses the probability of the trigram  $(w_i, w_{i-1}, w_{i-2})$  to predict the next word, while the bigram language model uses the probability of the bigram  $(w_i, w_{i-1})$  to predict the next word.

## THE PERPLEXITY OF N-GRAM MODELS

---

For an N-gram model, perplexity is calculated using the following generalized equation:

$$\text{perplexity} = \exp \left( -\frac{1}{N} \sum_{i=1}^N \log \left( \frac{\text{count}(w_i, w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})}{\text{count}(w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})} \right) \right) \quad (28)$$

where

*perplexity* is the perplexity of the language model

*N* is the number of words in the test set

$w_i$  is the *i*th word in the test set

$w_{i-1}$  is the  $(i - 1)$ th word in the test set

$w_{i-2}$  is the  $(i - 2)$ th word in the test set

$\text{count}(w_i, w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})$  is the number of times the n-gram

$(w_i, w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})$  appears in the training corpus

$\text{count}(w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})$  is the number of times the  $(n - 1)$ -gram

$(w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})$  appears in the training corpus

The perplexity equation can be used to evaluate the performance of an n-gram language model on a test set. The perplexity equation for an n-gram language model

## PERPLEXITY - EXAMPLE

---

Assuming we have estimated the probabilities of word sequences based on the MLE model, we can calculate the perplexity. The equation 28 can be rewritten as

$$\text{Perplexity} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}} \quad (29)$$

where  $N$  is the number of words in the sentence and  $P(w_1, w_2, \dots, w_N)$  is the probability of the entire sentence.

Let's calculate the perplexity for both a seen sentence ("I like cats") and an unseen sentence ("I like rabbits") based on the estimated probabilities from the MLE language model.

## PERPLEXITY - EXAMPLE

---

Assuming the estimated probabilities for the seen sentence "I like cats" are as follows:

$$P(\text{I}|\text{start}) = 0.25; P(\text{like}|\text{I}) = 0.5$$

$$P(\text{cats}|\text{like}) = 0.5; P(\text{end}|\text{cats}) = 0.05$$

$$P(\text{end}|\text{dogs}) = 0.05$$

We can calculate the probability of the sentences as follows:

$$\begin{aligned} P(\text{I like cats}) &= P(\text{I}|\text{start}) \cdot P(\text{like}|\text{I}) \cdot P(\text{cats}|\text{like}) \cdot P(\text{end}|\text{cats}) \\ &= 0.25 \cdot 0.5 \cdot 0.5 \cdot 0.05 = 0.003125 \end{aligned}$$

$$\begin{aligned} \text{Perplexity}_{\text{seen}} &= \sqrt{\frac{1}{P(\text{I like cats})}} \\ &= \sqrt{\frac{1}{0.003125}} = 17.89 \end{aligned}$$

## PERPLEXITY EXAMPLE

---

Now, let's calculate the perplexity for the unseen sentence "I like rabbits." Smoothing can be applied to "rabbits" as it is unseen in the training corpus. We can either use smoothing for all bigrams or use smoothing only for unseen word. In this case, we are using the smoothing only for the unseen tokens/words.

Assuming we use additive smoothing with  $\alpha = 0.5$  and a vocabulary size of  $V = 1000$ , we can estimate:

$$\begin{aligned} P_{\text{smooth}}(\text{rabbits}|\text{like}) &= \frac{\text{count}(\text{like}, \text{rabbits}) + \alpha}{\text{count}(\text{like}) + \alpha \cdot V} \\ &= \frac{0 + 0.5}{2 + 0.5 \cdot 1000} \\ &= 0.001 \end{aligned}$$

The probability of the unseen sentence "I like rabbits" using smoothing function can be calculated as:

$$P_{\text{smooth}}(\langle\text{end}\rangle|\text{rabbits}) = \frac{0 + 0.5}{2 + 0.5 \cdot 1000} = 0.001$$

$$\begin{aligned} P(\text{I like rabbits}) &= P(\text{I}|\text{start}) \cdot P(\text{like}|\text{I}) \cdot P_{\text{smooth}}(\text{rabbits}|\text{like}) \cdot P(\text{end}|\text{rabbits}) \\ &= 0.25 \cdot 0.5 \cdot 0.001 \cdot 0.001 \\ &= 1.25e - 07 \end{aligned}$$

The perplexity of this unseen sentence is

$$\begin{aligned} \text{Perplexity}_{\text{unseen}} &= \sqrt{\frac{1}{P(\text{I like rabbits})}} \\ &= \sqrt{\frac{1}{1.25e - 07}} \\ &= 2828.43 \end{aligned}$$

In conclusion, the perplexity for the seen sentence "I like cats" is approximately 17.89. However, the perplexity for the unseen sentence "I like rabbits" is very high (2828.43) as it contains an unseen word, *rabbit*. Perplexity serves as a metric to evaluate language models, where lower values indicate better performance in predicting given sentences.

Vocabulary size =  $V$ . The parameters for

- ▶ Unigram model -  $V$  parameters
- ▶ Bigram model -  $V^2$  parameters
- ▶ Trigram model -  $V^3$  parameters
- ▶ n-gram model -  $V^n$  parameters

The curse of dimensionality refers to the exponential increase in the size of the parameter space as the dimensionality of the data increases. In the context of an n-gram language model, the curse of dimensionality becomes apparent due to the rapidly growing number of parameters as the value of  $n$  increases.

As the free parameter space becomes larger, several challenges arise:

- ▶ **Data Sparsity** - Many n-grams will have zero or very low counts, making it difficult to reliably estimate their probabilities.
- ▶ **Over fitting** - The model may become overly sensitive to specific patterns in the training data, which may not generalize well to unseen data
- ▶ **Computational Complexity** - Training and inference become computationally expensive as the number of parameters grows

- ▶ Smoothing - Improves the estimation of probabilities for unseen n-grams
- ▶ Pruning - Low count n-grams can be removed to improve the estimation of probabilities and computational efficiency
- ▶ Back-off and interpolation - combines n-gram models of different orders to balance the trade-off between data sparsity and the complexity of the model



Ramaseshan Ramachandran  
Probabilistic Language Models