

# Probabilistic Language Models

Ramaseshan Ramachandran

.....

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

Markov Assumption  
Target and Context words  
Language Modeling using Unigrams  
Smoothing  
Bigram Language Model  
Bigram Language Model - Example

# INTRODUCTION

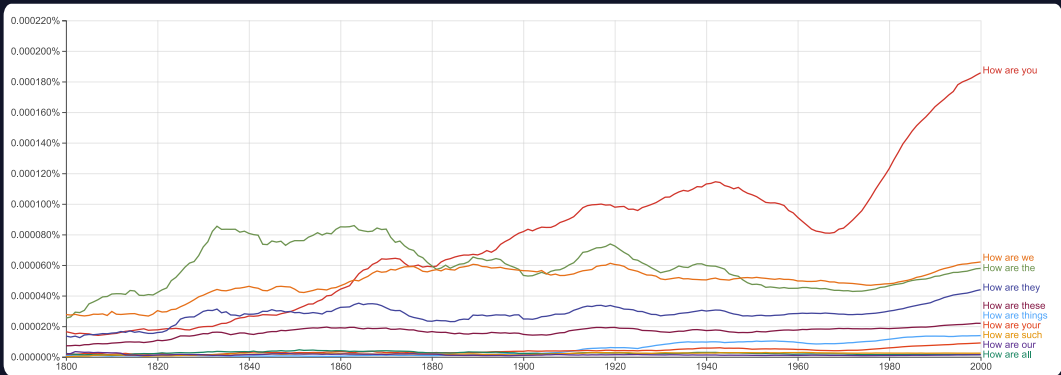
---

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

Ramaseshan

# INTRODUCTION

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



Source: Google NGram Viewer

# INTRODUCTION

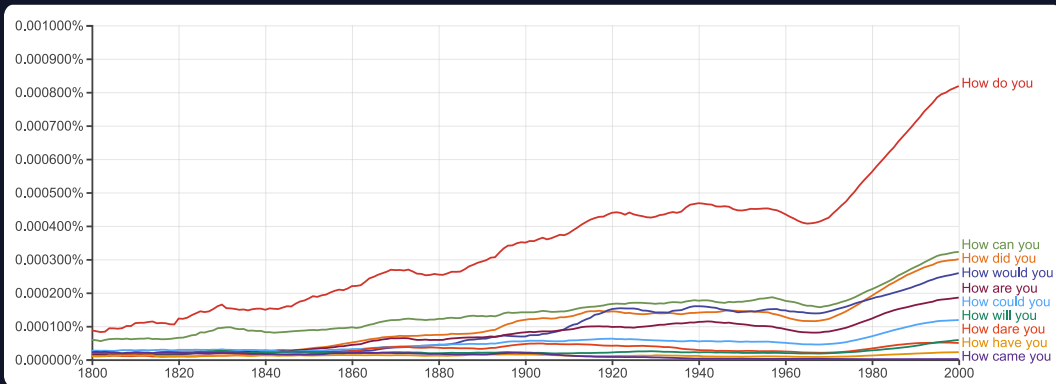
---

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

Ramaseshan

# INTRODUCTION

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



Source: Google NGram Viewer

# INTRODUCTION

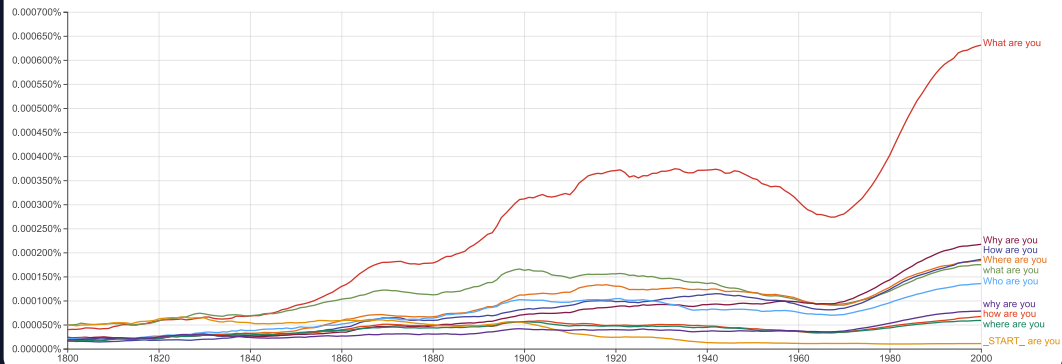
---

\_\_\_\_\_ are you?

Ramaseshan

# INTRODUCTION

\_\_\_\_\_ are you?



Source: Google NGram Viewer

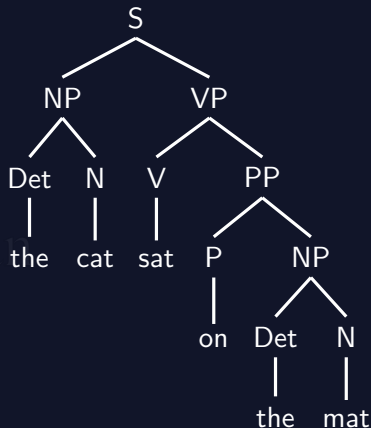


How do humans predict the next word?

- ▶ 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  $W$ .



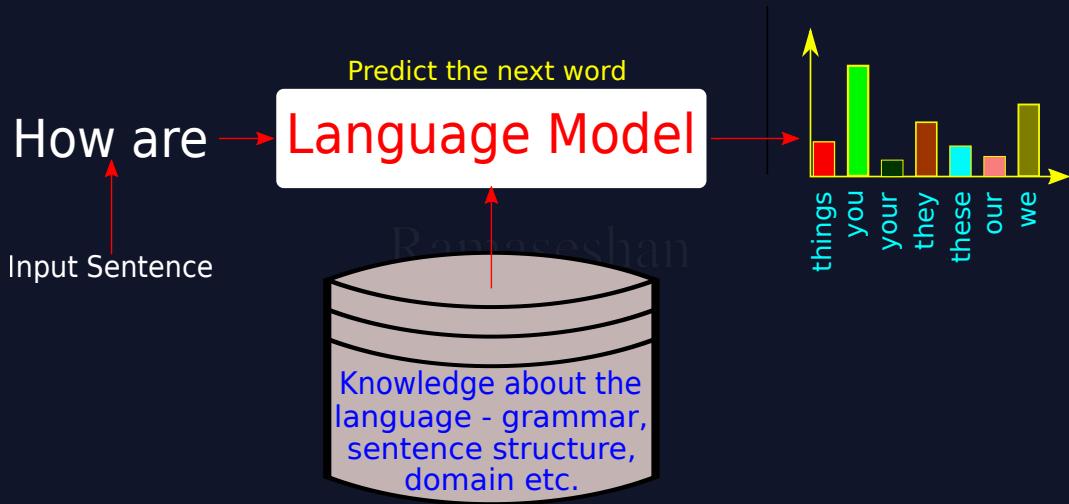
The current research in Language models focuses more on building the model from the huge corpus of text

# APPLICATIONS

---

Application	Sample Sentences
Speech Recognition	Did you hear <b>Recognize speech</b> or <b>Wreck a nice beach?</b>
Context sensitive Spelling	<b>One upon a tie, Their</b> lived aking
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

# PROBABILISTIC ASSERTIONS

---

In the logical assertions, we strictly rule out possibilities - if-else-endif

- ▶ Probabilistic assertions are about **possible worlds**
  - how probable the various worlds are and the set of all possible worlds is called the **sample space**

A probability model is a mathematical framework that describes the likelihood of different outcomes occurring in a certain event or situation. A probability model associates a numerical probability value with each possible world.

Let  $V$  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  $V$ . Let  $|V|$  denote the size of  $V$ .

Let  $W$  be infinite sequences of words from the collection of  $V$ . 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

$$\forall x \in W, p(x) \geq 0 \text{ and } \sum_{x \in W} p(X = x) = 1 \quad (1)$$

# PROBABILISTIC LANGUAGE MODEL

---

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

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) \quad (2)$$

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

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

A model which computes the probability for (2) or predicting the next word (3) 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  $P(\text{The cat meows})$



## CHAIN RULE

---

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.

# GENERATIVE MODEL

---

- ▶ A Generative model (GM) generates new sentences
- ▶ Produces sequences of words, sentences, or paragraphs that resemble any natural language
- ▶ Captures the underlying patterns and statistics of a corpus to create new sentences
- ▶ Learns the underlying structure of language - includes grammar, syntax, semantics(?), and context

# GENERATIVE LANGUAGE MODEL

---

- ▶ Ability to **synthesize** new sentences that are **statistically indistinguishable** from the training data (Ideal to pass the **Turing Test**)
- ▶ Describes the likelihood of any string (word or a sentence) using **probability distribution**
- ▶ Evaluates sentences - "The cat sat on the mat" is more likely than "The mat cat on the sat"
- ▶ Assists in predicting the next word or help in completing the sentence
- ▶ Provides alternate construct to a sentence
- ▶ Corrects spelling mistakes or grammar
- ▶ Provide translations or answer a question, if pairs of models are developed

Since natural languages are not like std-20 version of C++, the language models are not perfect :)

# GENERATIVE LANGUAGE MODEL

---

- ▶ Rely on large amounts of training data to learn the patterns and statistics
- ▶ Depends on the diversity of the training data to have a good quality output

# MATHEMATICAL DESCRIPTION OF A GENERATIVE MODEL

Here is the mathematical description of a generative model in NLP is the following:

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

where  $P(w_1, w_2, \dots, w_n)$  is the probability of the sequence of words  $w_1, w_2, \dots, w_n$  and  $P(w_i|w_1, w_2, \dots, w_{i-1})$  is the probability of the word  $w_i$  given the previous words, or its context  $w_1, w_2, \dots, w_{i-1}$

What is the probability of the sentence ***The cat sat on the mat?***

$$\begin{aligned} P(\textit{The cat sat on the mat}) = & P(\textit{The})P(\textit{cat}|\textit{The}) \times P(\textit{sat}|\textit{The cat}) \\ & \times P(\textit{on}|\textit{The cat sat}) \times P(\textit{the}|\textit{The cat sat on}) \\ & \times P(\textit{mat}|\textit{The cat sat on the}) \end{aligned} \quad (5)$$

The probability distribution of words in a language is used as the knowledge to generate new text that appears lexically similar to the text from the corpus.

## 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

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

Equation (6) can be generalized for an  $n$ -gram as

$$P(w_k | w_1^{k-1}) \approx P(w_k | w_{k-K+1}^{k-1}) \quad (7)$$

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

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) = P(w_1^n) \quad (8)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \dots, w_1) \quad (9)$$

$$= \prod_{k=1}^n P(w_k | w_1^{k-1}) \quad (10)$$

$$\approx \prod_{k=1}^n P(w_k | w_{k-K+1}^{k-1}) \quad (11)$$

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

- ▶ All words are generated independent of its history  $w_1, w_2, w_3, \dots, w_n$  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  $c_{w_i}$  is 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



## SMOOTHING

---

- ▶ 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:

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

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

# BIGRAM LANGUAGE MODEL

---

- ▶ 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, then they are time-invariant
- ▶ MLE picks up the word that is  $\frac{n_{w_c, w_i}}{n_{w_c}}$  where  $n_{w_c, w_i}$  is the number of times the words  $w_c, w_i$  occur together and  $n_{w_c}$  is the number of times the word  $w_c$  appears in the bigram sequence with any other word
- ▶ If  $V$  is the vocabulary of a corpus and  $V_c$  is the number of words in the vocabulary, what is the to number of parameters to be estimated?

The bigram probabilities are estimated using a COVID19 **corpus**.<sup>1</sup> The corpus was created by reading the abstracts of all the papers found in the corpus.

Total number of abstracts = 224672

The vocabulary count  $|V|$  = 249737

---

<sup>1</sup>[https://drive.google.com/file/d/1PVA5IR4bAn7RuIAeuSgxRBNMtmRGXaUm/view?usp=share\\_link](https://drive.google.com/file/d/1PVA5IR4bAn7RuIAeuSgxRBNMtmRGXaUm/view?usp=share_link)

# BIGRAM LANGUAGE MODEL - EXAMPLE

Context - $\langle s \rangle$	
$p(the \langle s \rangle)$	0.13
$p(in \langle s \rangle)$	0.05
$p(we \langle s \rangle)$	0.05
$p(this \langle s \rangle)$	0.03
$p(a \langle s \rangle)$	0.02
$p(however \langle s \rangle)$	0.02
$p(these \langle s \rangle)$	0.02
$p(to \langle s \rangle)$	0.01
$p(our \langle s \rangle)$	0.01
$p(it \langle s \rangle)$	0.01

Context - <b>we</b>	
$p(have we)$	0.04
$p(also we)$	0.04
$p(found we)$	0.04
$p(show we)$	0.03
$p(present we)$	0.03
$p(propose we)$	0.03
$p(report we)$	0.03
$p(describe we)$	0.02
$p(used we)$	0.02
$p(identified we)$	0.02

Context - <b>report</b>	
$p(the report)$	0.18
$p(a report)$	0.1
$p(of report)$	0.07
$p(on report)$	0.06
$p(that report)$	0.06
$p(we report)$	0.05
$p(here report)$	0.02
$p(. report)$	0.02
$p(describes report)$	0.02
$p(is report)$	0.02

The bigram probabilities are computed as follows:

$$P(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}w_n)}{\text{Count}(w_{n-1})} \quad (16)$$

$$P_L(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}w_n) + 1}{\text{Count}(w_{n-1}) + |V|} \quad (17)$$

where  $P_L$  is the Laplacian smoothing and  $|V|$  is the size of the vocabulary.

# 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 containing 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     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')
```

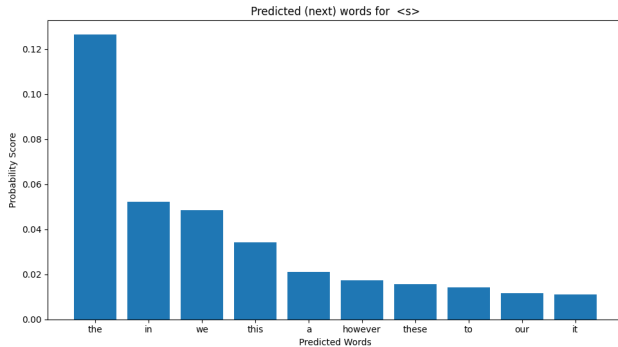
# MODEL PARAMETERS - BIGRAM EXAMPLE

```
65 <defaultdict, len() = 226126>
66 > special variables
67 > function variables
68 > '<s>': <defaultdict, len() = 27215>
69 > 'bovine': <defaultdict, len() = 209>
70 > 'viral': <defaultdict, len() = 1541>
71 > 'diarrhoea': <defaultdict, len() = 121>
72 > 'bvd': defaultdict(<function build_bigram_model.<locals>.  
73     > 'has': <defaultdict, len() = 1423>  
74     > special variables  
75     > function variables  
76     'been': 0.31419770372444694  
77     'made': 0.004620554466535984  
78     'shown': 0.013114907122187996  
79     'declared': 0.0014001680201624195  
80     'achieved': 0.0009334453467749463  
81     'proved': 0.0018668906935498926  
82     'consequences': 0.00023336133669373658  
83     'had': 0.00695416783347335  
84 Hold Alt key to switch to editor language hover
85 model = read_model()  
86
```

```
ct(lambda: collections.defaultdict(lambda: 0  
, "rb")) as openfile:  
  
l.load(openfile))  
(openfile)  
  
d19.txt')
```



# BIGRAM 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:

$$P(w_1, w_2, \dots, w_N) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \dots P(w_N|w_1, w_2, \dots, w_{N-1}) \quad (18)$$

### 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  $w_i$

$$P(w_i | w_1^{i-1}) = \frac{\text{count}(w_1^{i-1}, w_i)}{\text{count}(w_1^{i-1})} \quad (19)$$

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:

$$P(W) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \dots \cdot P(w_n|w_{n-1}) \quad (20)$$

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:

$$P(\text{"I love cats."}) = P(I) \cdot P(\text{love} | I) \cdot P(\text{cats} | \text{love}) \cdot P(. | \text{cats}) \quad (21)$$

$$P(\text{"Cats love playing."}) = P(\text{cats}) \cdot P(\text{love} | \text{cats}) \cdot P(\text{playing} | \text{love}) \cdot P(. | \text{playing}) \quad (22)$$

$$P(\text{"Dogs chase cats."}) = P(\text{dogs}) \cdot P(\text{chase} | \text{dogs}) \cdot P(\text{cats} | \text{chase}) \cdot P(. | \text{cats}) \quad (23)$$

## 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:

$$P(I) = \frac{\text{Count}(I)}{\text{Total number of words in the corpus}} \quad (24)$$

$$P(\text{love} | I) = \frac{\text{Count}(\text{love} | I)}{\text{Count}(I)} \quad (25)$$

$$P(\text{cats} | \text{love}) = \frac{\text{Count}(\text{cats} | \text{love})}{\text{Count}(\text{love})} \quad (26)$$

$$P(. | \text{cats}) = \frac{\text{Count}(. | \text{cats})}{\text{Count}(\text{cats})} \quad (27)$$

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:

$$\begin{aligned} \text{perplexity} &= \exp \left( -\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_{i-1}) \right) \\ &= \exp \left( -\frac{1}{N} \sum_{i=1}^N \log \left( \frac{\text{count}(w_i, w_{i-1})}{\text{count}(w_{i-1})} \right) \right) \end{aligned} \quad (28)$$

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:

$$perplexity = \exp \left( -\frac{1}{N} \sum_{i=1}^N \log \left( \frac{count(w_i, w_{i-1}, w_{i-2})}{count(w_{i-1}, w_{i-2})} \right) \right) \quad (29)$$

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

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

$count(w_{i-1}, w_{i-2})$  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:

$$perplexity = \exp \left( -\frac{1}{N} \sum_{i=1}^N \log \left( \frac{count(w_i, w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})}{count(w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})} \right) \right) \quad (30)$$

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

$count(w_i, w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})$  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

$count(w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})$  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

Assuming we have estimated the probabilities of word sequences based on the MLE model, we can calculate the perplexity. The equation 30 can be rewritten as

$$\text{Perplexity} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}} \quad (31)$$

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:

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

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

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

We can calculate the probability of the sentences as follows:

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

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

## 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:

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

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

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.



# CURSE OF DIMENSIONALITY

---

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