

Probabilistic Language Models

n-grams, Smoothing and Perplexity

Ramaseshan Ramachandran

①

②

③

④

⑤

⑥

⑦

⑧

⑨

⑩

⑪

⑫

⑬

GUESS THE MISSING WORD I

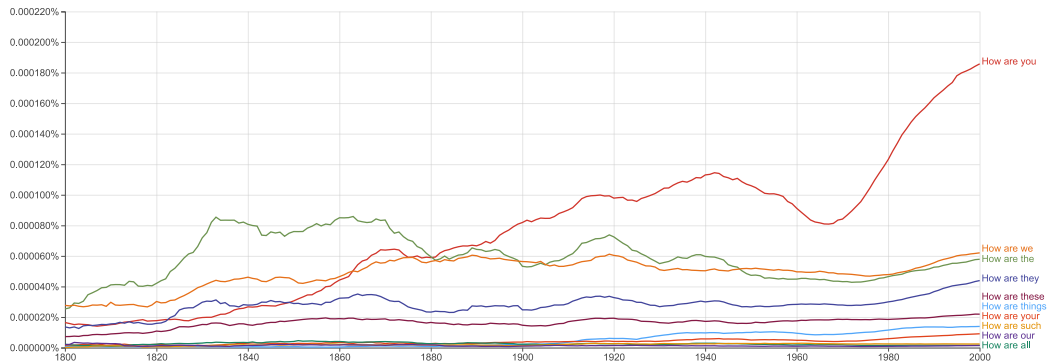
How are _____

Can you guess the missing word?

GUESS THE MISSING WORD I

How are ____

Can you guess the missing word?



Source: Google NGram Viewer

GUESS THE MISSING WORD II

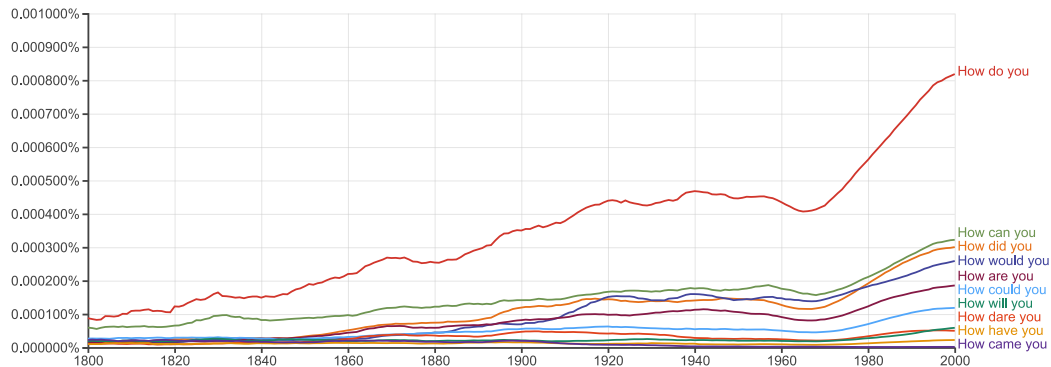
How ____ you

Can you guess the missing word?

GUESS THE MISSING WORD II

How ____ you

Can you guess the missing word?



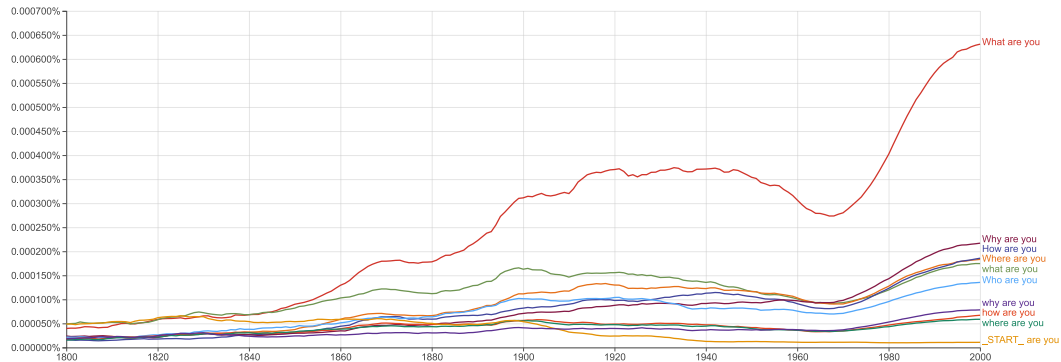
Source: Google NGram Viewer

GUESS THE MISSING WORD III

_____ are you

GUESS THE MISSING WORD III

_____ are you



Source: Google NGram Viewer

HOW DO WE PREDICT THE NEXT WORD?

How do humans predict the next word?

- ▶ Domain knowledge
- ▶ Syntactic knowledge
- ▶ Lexical knowledge
- ▶ Knowledge about the sentence structure
- ▶ 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

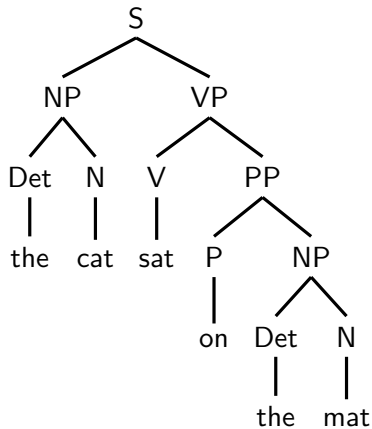
LEXICAL KNOWLEDGE

Lexical knowledge encompasses the nature of language, specifically the words and their meanings. Words are acquired through associative learning, such as collocations and expressions, without explicitly comprehending their underlying constituents.



SYNTACTIC RULES

We analyze the sentence structure to understand the grammatical role needed for the missing word



SEMANTIC CUES

We use our understanding of word meaning and context to infer the most likely word that fits the overall meaning of the sentence

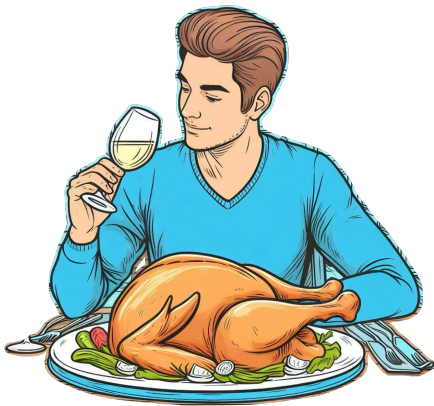
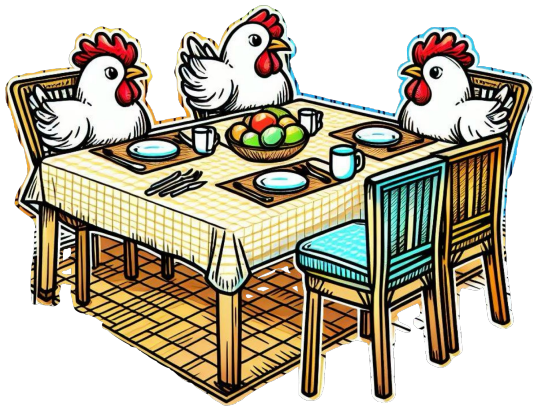


WORLD KNOWLEDGE

We draw on our knowledge of the world to make educated guesses about the missing word.

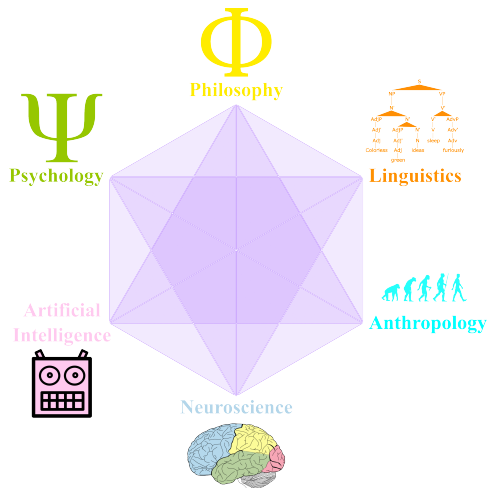


PRAGMATIC CONSIDERATIONS



Chicken is ready to eat

COGNITIVE SCIENCE



- Knowledge is available as types, relations and their complex relationship - represented in the form of first-order and propositional logic

Who is an all rounder in cricket?

$\exists x(\text{Bats well}(x) \wedge \text{Fields well}(x) \wedge \text{Bowls well}(x)) \Rightarrow \text{All Rounder}(x)$

The facts are hard facts - there could be no uncertainty

PROBABILISTIC REASONING

- ▶ Knowledge is available in the form of parametric relationships - co-occurrences, correlations, incidence matrix, conditional relationships
- ▶ Handles uncertainty well using the observed parameters
- ▶ $\exists w_i \exists w_j (\text{Similar Contexts}(x) \wedge \text{Similar Contexts}(y)) \rightsquigarrow \text{Similar}(x, y)$
- ▶ How are they related in a corpus?
- ▶ Context words may differ and not fixed

WHY PROBABILITY? I

- ▶ Provides methods to predict the next word in a sequence based on sampled data.
- ▶ Makes informed decisions when there's uncertainty and observed data.
- ▶ Quantifies the chances or likelihoods associated with various outcomes.
- ▶ Probability of a sentence.
- ▶ Probability of the next word in a sentence - how likely to predict you as the next word.
- ▶ Likelihood of the next word is formalized by counting words in a document.
- ▶ Probability is the likelihood that an event will occur.
For example, a fair coin has a 50% chance of landing on heads.
- ▶ Probabilities can be expressed as percentages, decimals, or fractions.

DISCRETE SAMPLE SPACE

Consider following text (count of words = 15)

A weather balloon, currently at a fixed altitude above Earth, releases a collection of instruments.

Experiment - Extracting tokens from a document

Outcome - Every token/word(x) in the document

Token	Count	Probability
a	3	0.20
weather	1	0.07
balloon,	1	0.07
currently	1	0.07
at	1	0.07
fixed	1	0.07
altitude	1	0.07
above	1	0.07
earth,	1	0.07
releases	1	0.07
collection	1	0.07
of	1	0.07
instruments.	1	0.07

DISCRETE SAMPLE SPACE

- ▶ The outcome of the experiment - 54 samples (words). They constitute the sample space, Ω or the set of all possible outcomes
- ▶ Each word in this sample belongs to Ω , represented by $x \in \Omega$
- ▶ Each sample $x \in \Omega$ is assigned a probability score $[0, 1]$
- ▶ A probability function or probability distribution function distributes the probability mass of 1 to the all the samples in the sample space Ω such that
- ▶ All the words in the Ω , must satisfy the following constraints:
 1. $p(x) \in [0, 1], \forall x \in \Omega$ and
 2. $\sum_{x \in \Omega} p(x) = 1$

EVENTS

An event is a collection of samples of the same type.

$$p(E) = \sum_{x \in \Omega} p(x) \quad (1)$$

'a', 'weather', 'balloon', 'is', 'floating',
'at', 'a', 'constant', 'height', 'above',
'earth', 'when', 'it', 'releases', 'a', 'pack',
'of', 'instruments', 'level', 'a', 'if', 'the',
'pack', 'hits', 'the', 'ground', 'with', 'a',
'downward', 'velocity', 'of', 'm', 's', 'how',
'far', 'did', 'the', 'pack', 'fall', 'b',
'calculate', 'the', 'distance', 'the', 'ball',
'has', 'rolled', 'at', 'the', 'end', 'of', 's'

Events can be described as a variable
taking a certain value

In the Bag of words (BOW), the word
type **the** occurs 6 times. Then, $E_{\text{the}} = 6$

$$P(E_{\text{the}}) = 6 \times \frac{1}{52} = 0.115$$

In the BOW, the word type **pack** occurs 3
times. Then, $E_{\text{pack}} = 3$

$$p(E_{\text{pack}}) = 3 \times \frac{1}{52} = 0.058$$

RANDOM VARIABLE

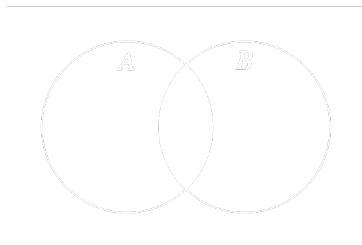
- ▶ A random variable is a variable whose possible values are numerical outcomes of a random phenomenon.
- ▶ Two types - continuous and discrete - for NLP, they are discrete
- ▶ To capture the type-token distinction, we use random variable W . $W(x)$ maps to the sample $x \in \Omega$. V is the set of **types or vocabulary** and the value is represented by a variable v .
- ▶ Given a random variable V and a value v , $p(V = v)$ is the probability of the event that V takes the value v , i.e.: $p(V = v) = P(x \in \Omega : V(x) = v)$
- ▶ $p(V = \text{'the'}) = p(\text{'the'}) = 0.115$
- ▶ Random variables are useful in describing/constructing various events

JOINT PROBABILITY

Given any two events E_1 and E_2 , the probability of their conjunction

$$p(E_1, E_2) = p(E_1 \cap E_2) \quad (2)$$

is called the **joint probability** of E_1 and E_2 . This probability, E_1 and E_2 , occurs simultaneously.



Example

The probability of the first letter of **t** and the second letter **h** is $p(F = 't', S = 'h')$. If the sample space is only the word **the** then the probability of $p(F = 't', S = 'h')$ is 1. If the sample space is the entire English language, then the probability would be very small, but still larger than the probability of the entire word **the**.

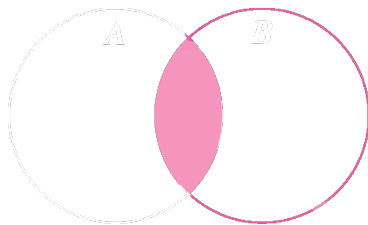
CONDITIONAL PROBABILITY

When we have partial knowledge influencing the outcome of an experiment, we use it to update the outcome.

The conditional probability $p(E_2|E_1)$ is the probability of event E_2 given that event E_1 has occurred. $P(E_2|E_1)$ is defined as:

$$p(E_1|E_2) = \frac{p(E_1, E_2)}{p(E_2)}, \text{ if } p(E_2) > 0 \quad (3)$$

$$= \frac{E_1 \cap E_2}{p(E_2)} \quad (4)$$



$p(A)$ = size of A relative to Ω

$p(A, B)$ = size of $A \cap B$ relative to Ω

$P(A|B)$ = size of $A \cap B$ relative to B

CONDITIONAL PROBABILITY - BI-GRAM EXAMPLE

Let consider a corpus of Kinematics problems in physics (very small corpus).

- ▶ Bigram Sample Space - $w_1, w_2 \in \Omega = 3767$
- ▶ $A = \{w_1, w_2\} = \{\text{average}, \star\}$ - bigram starting with the word average
- ▶ $B = \{w_1, w_2\} = \{\star, \text{speed}\}$ - bigram ending with the word speed

Let $p(\text{average}) = 0.036$, $p(\text{speed}) = 0.114$ and $p(\text{average}, \text{speed}) = 0.004$.

Then

$$p(\text{speed}|\text{average}) = \frac{0.004}{0.036} = 0.111$$

$$p(\text{average}|\text{speed}) = \frac{0.004}{0.114} = 0.035$$

CONDITIONAL PROBABILITY - TRIGRAM EXAMPLE I

Let consider a corpus of Kinematics problems in physics that contains about 280+ problems (very small corpus).

- ▶ Trigram Sample Space - $\{(w_1, w_2), w_3\} \in \Omega = 5902$
- ▶ $A = \{(w_1, w_2), w_3\} = \{\text{average, speed, of}\}$ - trigram starting with (average, speed)
- ▶ $B = \{(w_1, w_2), w_3\} = \{(\text{average, speed}), \text{of}\}$ - trigram ending with the word **of**
- ▶ $C = \{(w_1, w_2), w_3\} = \{(\text{average, speed}), \text{for}\}$ - trigram ending with **for**
- ▶ $D = \{(w_1, w_2), w_3\} = \{(\text{average, speed}), \text{during}\}$ - trigram ending with the word **during**

CONDITIONAL PROBABILITY - TRIGRAM EXAMPLE II

- ▶ $p(\text{average, speed}) = 0.0032$; $p(\text{average, speed, of}) = 0.0007$
- ▶ $p(\text{average, speed, for}) = 0.0005$; $p(\text{average, speed, during}) = 0.0002$
- ▶ $p(\text{of}|\text{average, speed}) = \frac{0.0007}{0.0032} = 0.21875$
- ▶ $p(\text{for}|(\text{average, speed})) = \frac{0.0005}{0.0032} = 0.15525$
- ▶ $p(\text{during}|(\text{average, speed})) = \frac{0.0002}{0.0032} = 0.0625$

Break

A model that captures syntax and semantic patterns to predict the likelihood of a word

THE LANGUAGE MODEL

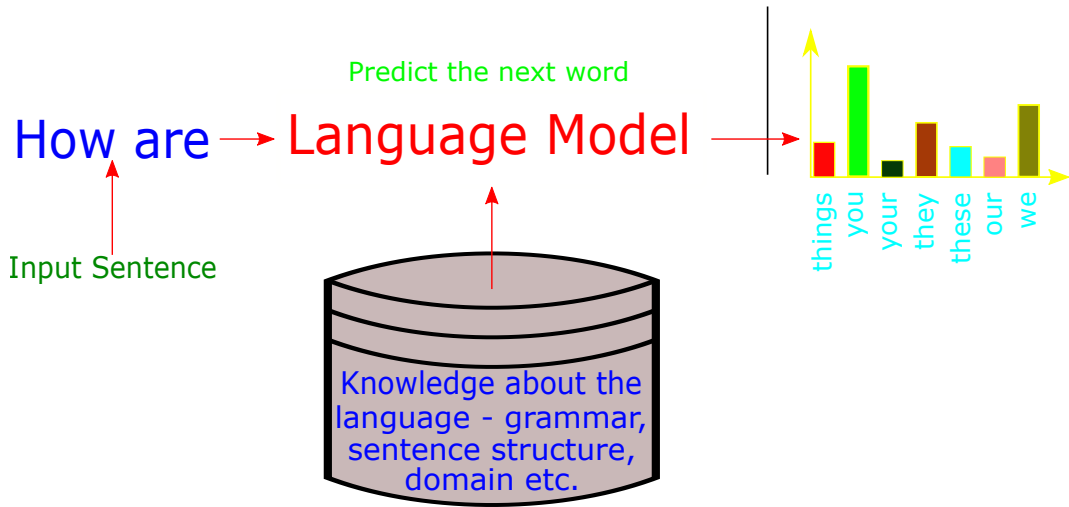
- ▶ Natural language sentences can be described by parse trees which use the morphology of words, syntax and semantics
- ▶ Probabilistic thinking - finding how likely a sentence occurs or formed, given the word sequence.
- ▶ In probabilistic world, the Language model is used to assign a probability $P(W)$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

- ▶ **Uncertainty and stochasticity** can arise from many sources
 - Speech recognition systems cannot depend on the processed speech signals.
 - Context understanding is important to transcribe incoherent speech signals
- ▶ **No single method** exists for generating natural language or performing translation
 - Almost for every given word, there are multiple possible **next words**
- ▶ **Incomplete observability**
 - Have we captured all possible English sentences in the corpus?
- ▶ **Incomplete modeling**
 - Use a **simple rule** rather than a complex one but make sure it is deterministic one
- ▶ How do **model systems** that handle **Inherent stochasticity**?
- ▶ We need models to **represent and reason** under conditions of uncertainty
- ▶ **Propositional Logic** provides a set of formal rules to determining whether a proposition is **true or false**

WHY PROBABILISTIC MODEL? II

- ▶ **Probability theory** offers **formal rules** to determine the **likelihood of a proposition** based on the likelihood of others
 - ▶ It provide a powerful framework for reasoning under uncertainty
-
- ▶ It enables the model to predict the most probable next set of words for a given word or phrase, enhancing applications like text generation, machine translation, and speech recognition
 - ▶ It moves beyond rigid rule-based systems to more **adaptive and context-aware** mechanisms
 - ▶ A probabilistic model continuously assesses the rank of words in a sequence, phrase, or sentence based on their frequency of occurrence.
-

WHY PROBABILISTIC MODEL? III

- ▶ Identify the most likely next set of words for a given word
- ▶ Determine the probability of the next word given the previous word using conditional probability - $p(\text{are}|\text{how}) = \frac{p(\text{how}, \text{are})}{p(\text{how})}$

LANGUAGE MODELS

We want the language models to answer

- ▶ How likely is the following word?
- ▶ How probable is it that a sequence of English words constitutes good English?
- ▶ How probable is the second sentence, given that the preceding sentence is contextually relevant?
- ▶ How likely is it that the word selected is the most appropriate choice?
- ▶ How probable is it that the sentence is considered fluent in terms of syntax and semantics?

$p(\text{the cat sat on the mat}) > p(\text{the mat sat on the cat})$

- ▶ 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 probabilistic language model predicts the likelihood of word sequences (***n-gram***) in natural language using probability theory

LANGUAGE MODEL DEFINITION

- ▶ A probability model describes the likelihood of outcomes in an event.
- ▶ Let V be a finite vocabulary, $\triangleleft$ and $\triangleright$ be start and stop symbols, and $|V|$ denote the size of V .
- ▶ Let W be the set of all finite sequences of words from V , starting with $\triangleleft$ and ending with $\triangleright$. A language model is a probability distribution over a random variable X taking values in W .

Or $p: W \rightarrow \mathbb{R}$ such that

$$\forall x \in W, p(x) \geq 0 \text{ and } \sum_{x \in W} p(X = x) = 1 \quad (5)$$

PROBABILISTIC LANGUAGE MODEL

Goal: Compute the probability of a sequence of words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) \quad (6)$$

Task: Given the context, find the next word using

$$P(w_n | w_1, w_2, w_3, \dots, w_{n-1}) \quad (7)$$

A model which computes the probability (6), predicts the next word (7), or completes a partial sentence is called a Probabilistic Language Model.

The goal is to learn the joint probability function of sequences of words in a language. Ideally, 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.

The probability of the sentence I am a large language model is the product of the probabilities of the individual words I, am, a, large, language, and model.

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

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= p(w_1) \cdot p(w_2|w_1) \cdot \dots \cdot p(w_n|w_1, w_2, \dots, w_{n-1}) \\ &= \prod_{i=1}^n p(w_i|w_{1:i-1}) \end{aligned}$$

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

CHAIN RULE - EXAMPLE

Any sentence can be decomposed using the chain rule of probability (conditioned on the context¹) as

$$\begin{aligned}P(w_1, \dots, w_n) &= p(w_1)p(w_2|w_1) \cdots p(w_n|w_{1:n-1}) \\&= \prod_{i=1}^n p(w_i|w_{1:i-1})\end{aligned}$$

The probability of the sentence <start> I am a large language model <end> is

$$\begin{aligned}P(\text{sentence}) &= p(I | \langle s \rangle) \\&\quad \times p(\text{am} | \langle s \rangle, I) \times p(a | \langle s \rangle, I, \text{am}) \times \cdots \\&\quad \times p(\langle /s \rangle | \langle s \rangle, I, \text{am}, a, \text{large}, \text{language}, \text{model})\end{aligned}\tag{8}$$

¹The history of the words that occurred before the current word

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

- ▶ Has the ability to synthesize new sentences that are statistically indistinguishable from those in the training data (an ideal condition, akin to passing the Turing Test).
- ▶ Defines a probability distribution over words and sequences, describing the likelihood of different sentences.
- ▶ Evaluates sentences. "The cat sat on the mat" is more probable than "The mat cat on the sat."
- ▶ Predicts the next word or completes a partially written sentence.
- ▶ Suggests alternative sentence formulations.
- ▶ Corrects spelling or grammatical errors.
- ▶ Provides translations or answers questions, if trained on appropriate paired data.

Since natural languages are not deterministic like the C++ standard, language models will never be 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

GENERATIVE MODEL

A generative model in NLP is described as:

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

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?***

$$\begin{aligned} P(\text{The cat sat on the mat}) = & p(\text{The})P(\text{cat}|\text{The}) \times p(\text{sat}|\text{The cat}) \\ & \times p(\text{on}|\text{The cat sat}) \times p(\text{the}|\text{The cat sat on}) \\ & \times p(\text{mat}|\text{The cat sat on the}) \end{aligned} \quad (10)$$

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 I

Markov Assumption: The future behavior of a dynamic system depends on its **recent history** and not on the entire history. Only last k words are included in history (older words are irrelevant) - k^{th} order Markov model

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

$$P(w_k | w_{1:k-1}) \approx P(w_k | w_{k-1}) \quad (11)$$

Equation (11) can be generalized for an n -gram as

$$P(w_k | w_{1:k-1}) \approx p(w_k | w_{k-K+1:k-1}) \quad (12)$$

where K is the size of the n -gram.

MARKOV ASSUMPTION II

The joint probability of a sequence can be re-written as

$$P(W) = p(w_1, w_2, w_3, \dots, w_n) = p(w_{1:n}) \quad (13)$$

$$= p(w_1)p(w_2|w_1)p(w_3|w_2, w_1) \dots p(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \dots, w_1) \quad (14)$$

$$= \prod_{k=1}^n p(w_k | w_{1:k-1}) \quad (15)$$

$$\approx \prod_{k=1}^n p(w_k | w_{k-K+1:k-1}) \quad (16)$$

MAXIMUM LIKELIHOOD ESTIMATION

Given a training corpus of sentences, we can calculate the MLE for 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 for probability $p(w_i|w_{1:i-1})$ is estimated by counting the occurrences of w_i in the given context and normalizing by the total count of that context.

$\text{count}(w_{1:i-1}, w_i)$ represents the number of times the context $w_{1:i-1}$ is followed by w_i , while $\text{count}(w_{1:i-1})$ is the count of the context itself.

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

```
sentence_ngrams = list(ngrams(word_tokenize(sentence.lower()),  
                             n=ngram_count,  
                             pad_left=True,  
                             pad_right=True,  
                             left_pad_symbol='<start>'  
                             right_pad_symbol='<end>'))
```

NLTK function to pad start and end
of a sentence with symbols



MLE FOR A TRIGRAM EXAMPLE

Given the following counts from a training corpus:

- ▶ $\text{Count}(\text{"the cat sat"}) = 3$
- ▶ $\text{Count}(\text{"cat sat on"}) = 2$
- ▶ $\text{Count}(\text{"sat on the"}) = 2$
- ▶ $\text{Count}(\text{"on the mat"}) = 3$
- ▶ $\text{Count}(\text{"the cat"}) = 5$
- ▶ $\text{Count}(\text{"cat sat"}) = 4$
- ▶ $\text{Count}(\text{"sat on"}) = 5$
- ▶ $\text{Count}(\text{"on the"}) = 6$

Compute the trigram probability:

$$p(\text{"the"} \mid \text{"sat on"}) = \frac{\text{Count}(\text{"sat on the"})}{\text{Count}(\text{"sat on"})} = \frac{2}{5} = 0.4$$

Similarly,

$$p(\text{"mat"} \mid \text{"on the"}) = \frac{\text{Count}(\text{"on the mat"})}{\text{Count}(\text{"on the"})} = \frac{3}{6} = 0.5$$

PITFALLS IN USING MLE FOR LANGUAGE MODELS

1. Zero Probability - MLE assigns zero probability to unseen words or n-grams in training data, causing issues during testing or deployment
2. Overfitting to Training Data - MLE maximizes the probability of observed data, it can overfit to frequently seen patterns, failing to generalize well to less common or new sequences, resulting in poor model robustness and creativity
3. Bias Toward Frequent Patterns - Limiting diversity in models can cause them to ignore rare or minority cases, reducing the diversity and innovation of the generated text and amplifying data biases.

UNKNOWN WORDS

- ▶ 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 probability will be used for computation
- ▶ This eliminates zero probability for sentences containing OOV

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

$$p(w_i) = \frac{C_{w_i} + \alpha}{C_W + \alpha|V|} \quad (18)$$

If $\alpha = 1$, this is called Laplace (add-one) smoothing:

$$p(w_i) = \frac{C_{w_i} + 1}{C_W + |V|} \quad (19)$$

LANGUAGE MODELING USING UNIGRAMS

- ▶ Every word $w_1, w_2, w_3, \dots, w_n$ is generated independently of its history. None of them depends on the others
- ▶ Not a good model for language generation
- ▶ It has $|V|$ 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 tokens in the corpus
- ▶ It cannot pick up the regularities present in the corpus
- ▶ It is more likely to generate **the the the the** as a sentence than a grammatically valid sentence

BIGRAM LANGUAGE MODEL

- ▶ 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:

$$P(\mathbf{W}) = \prod_{i=1}^{n+1} p(w_i | w_{i-1}) \quad (20)$$

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
- ▶ The next word is selected based on $\frac{n_{w_c, w_i}}{n_{w_c}}$
 - ▶ n_{w_c, w_i} is the number of times the words w_c, w_i occur together
 - ▶ n_{w_c} is the number of times the word/context w_c appears in the bigram sequence with any other word
- ▶ If V is the vocabulary of a corpus and $|V|$ is the number of words in the vocabulary, what is the to number of parameters to be estimated?

BIGRAM LANGUAGE MODEL

Let's define the Bigram Language Model equation:

$$P(W) = p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_2) \cdot \dots \cdot p(w_n|w_{n-1}) \quad (21)$$

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 follows²:

$$P(\text{"I love cats."}) = p(I) \cdot p(\text{love} \mid I) \cdot P(\text{cats} \mid \text{love}) \cdot p(. \mid \text{cats}) \quad (22)$$

$$P(\text{"Cats love playing."}) = p(\text{cats}) \cdot p(\text{love} \mid \text{cats}) \cdot P(\text{playing} \mid \text{love}) \cdot p(. \mid \text{playing}) \quad (23)$$

$$P(\text{"Dogs chase cats."}) = p(\text{dogs}) \cdot p(\text{chase} \mid \text{dogs}) \cdot p(\text{cats} \mid \text{chase}) \cdot p(. \mid \text{cats}) \quad (24)$$

²Omitting $\triangleleft$ and $\triangleright$ for the sake of compact representation. In real computation, they should be included

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.":

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

We calculate the probabilities as follows:

$$p(I) = \frac{\text{Count}(I)}{\text{Total number of words in the corpus}} \quad (25)$$

$$p(\text{love} | I) = \frac{\text{Count}(\text{love} | I)}{\text{Count}(I)} \quad (26)$$

$$p(\text{cats} | \text{love}) = \frac{\text{Count}(\text{cats} | \text{love})}{\text{Count}(\text{love})} \quad (27)$$

$$p(. | \text{cats}) = \frac{\text{Count}(. | \text{cats})}{\text{Count}(\text{cats})} \quad (28)$$

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

BIGRAM LANGUAGE MODEL - EXAMPLE

Context - <i><s></i>	
p(the <s>)	0.13
p(in <s>)	0.05
p(we <s>)	0.05
p(this <s>)	0.03
p(a <s>)	0.02
p(however <s>)	0.02
p(these <s>)	0.02
p(to <s>)	0.01
p(our <s>)	0.01
p(it <s>)	0.01

Context - <i>we</i>	
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 - <i>report</i>	
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:

$$p(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}w_n)}{\text{Count}(w_{n-1})} \quad (29)$$

$$p_L(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}w_n) + 1}{\text{Count}(w_{n-1}) + |V|} \quad (30)$$

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

BUILDING A BIGRAM MODEL - CODE I

```
#compute the bigram model
def build_bigram_model():
    bigram_model = collections.defaultdict(
        lambda: collections.defaultdict(lambda: 0))
    for sentence in kinematics_corpus.sents():
        sentence = [word.lower() for word in sentence
                     if word.isalpha()] # get alpha only
        #Collect all bigrams counts for (w1,w2)
        for w1, w2 in bigrams(sentence):
            bigram_model[w1][w2] += 1
        #compute the probability for the bigram containing w1
        for w1 in bigram_model:
            #total count of bigrams containing w1
            total_count = float(sum(bigram_model[w1].values()))
            #distribute the probability mass for all bigrams starting with w1
            for w2 in bigram_model[w1]:
                bigram_model[w1][w2] /= total_count
```

BUILDING A BIGRAM MODEL - CODE II

```
    return bigram_model

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')
```

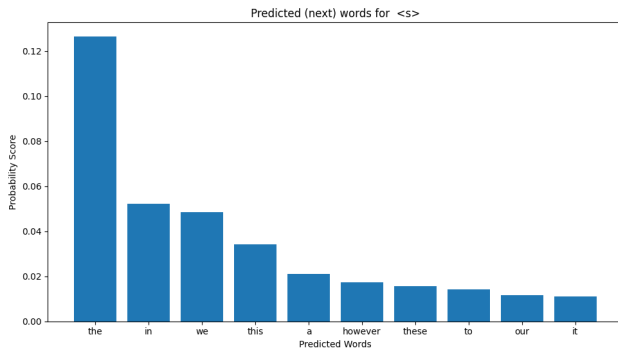
BUILDING A BIGRAM MODEL - CODE III

```
plt.xlabel('Predicted Words')
plt.title('Predicted words for ' + first_word)
plt.show()

if __name__ == '__main__':
    predict_next_word('how')
```

```
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  
'proved': 0.0018668906935498926  
'consequences': 0.00023336133669373658  
'had': 0.00695416783347335
```

BIGRAM MODEL - NEXT WORD PREDICTION



BUILDING N(5)GRAM LANGUAGE MODEL - SAMPLE PROBABILITIES

Building a sentence with the context- [<start>, <start>, <start>, <start>, industrial]
with a probability of 0.00081

Word	probability
production	0.5278 ①
growth	0.0833
restructuring	0.0556
has	0.2105 ②
in	0.1579
also	0.1053
continued	0.5000 ③
been	0.2500
shown	0.2500
to	1.0000 ④

Word	probability
expand	0.6667 ⑤
rise	0.3333
at	0.5000 ⑥
,	0.5000
a	1.0000 ⑦
most	1.0000 ⑧
satisfactory	1.0000 ⑨
rate	1.0000 ⑩
.	1.0000 ⑪
<end>	1.0000 ⑫

Sentence : Industrial production has continued to expand at a most satisfactory rate.

Probability of the sentence = 1.5033637764498063e-05

NUMBER OF PARAMETERS TO ESTIMATE

Let V be the vocabulary of a corpus and $|V|$ its size.

► **Unigram model:**

The number of parameters is the probability for each word in the vocabulary:

$$\text{Parameters} = |V|$$

► **n -gram model:**

Probabilities of each word conditioned on the previous $n - 1$ words. For each history of length $n - 1$, there are $|V| - 1$ free parameters:

$$\text{Parameters} \approx |V|^{n-1} \times (|V| - 1)$$

where $|V|^{n-1}$ is the number of possible histories of length $n - 1$.

CURSE OF DIMENSIONALITY

- ▶ The curse of dimensionality refers to the exponential increase in the size of the parameter space as the dimensionality of the data increases.
 - ▶ A fundamental problem that makes language modeling and other learning problems difficult
- ▶ Unigram model $\approx |V|$ parameters
 - ▶ Bigram model $\approx |V|^2$ parameters
 - ▶ Trigram model $\approx |V|^3$ parameters
 - ▶ n-gram model $\approx |V|^n$ parameters
 - ▶ If one wants to estimate the joint probability distribution of 10 words in a language with a million words as vocabulary, then we need to estimate
 - ▶ $|V|^{n-1} \times (|V| - 1) = (10^6)^9 \times (10^6 - 1) \approx 10^{60}$ parameters

CONVENTIONS FIRST: PADDING, STOP SYMBOL, UNKNOWNNS

Before any two n-gram probabilities can be compared, the bookkeeping must match. The standard conventions (used throughout this course and in Assignment 4):

- ▶ Pad each sentence with $(n-1)$ start markers $\langle s \rangle$ and **one** stop marker $\langle /s \rangle$. The stop marker is a real event the model must predict, or probabilities over sentence lengths don't sum to 1
- ▶ The vocabulary V includes $\langle /s \rangle$ and $\langle \text{unk} \rangle$; any word unseen in training maps to $\langle \text{unk} \rangle$
- ▶ Probability (and perplexity) is computed over the *predicted* tokens: the words **plus** $\langle /s \rangle$, never the start padding

Worked micro-example (bigram, add-1, train = {"the cat sat", "the cat ran", "the dog sat"}):

- ▶ MLE: $p(\text{cat} \mid \text{the}) = \frac{2}{3}$ (counts 2 of 3 continuations of *the*)
- ▶ Add-1 with $|V| = 7$ (five words + $\langle /s \rangle$ + $\langle \text{unk} \rangle$): $p(\text{cat} \mid \text{the}) = \frac{2+1}{3+7} = \frac{3}{10}$

Same corpus, same query, yet two different numbers. *State your conventions.*

PERPLEXITY

A reliable language model assigns a high probability to text written in genuine English. This property can be quantified using *cross-entropy*. The average negative log-likelihood per word is computed as

$$H(W) = -\frac{1}{n} \sum_{i=1}^n \log p(w_i | w_{1:i-1}) \quad (31)$$

$$\text{Perplexity}(W) = 2^{H(W)} \quad (32)$$

- ▶ Perplexity is the *geometric mean branching factor*³ of the model; it can be interpreted as the average number of word choices the model is perplexed by at each prediction
- ▶ Measures how well a language model predicts the next word in a sequence
- ▶ Lower perplexity indicates a better language model
- ▶ If a language models perplexity is 100, its as uncertain as choosing a random word from 100 options on average.

³refers to the average number of plausible next-word choices the model considers at each step, measured using the geometric mean

PERPLEXITY OF A BIGRAM MODEL

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

$$\begin{aligned} PP &= \exp \left(-\frac{1}{N} \sum_{i=1}^N \log p(w_i | w_{i-1}) \right) \\ &= \exp \left(-\frac{1}{N} \sum_{i=1}^N \log \left(\frac{\text{count}(w_i, w_{i-1})}{\text{count}(w_{i-1})} \right) \right) \end{aligned} \quad (33)$$

where

PP is the perplexity 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

PERPLEXITY OF A TRIGRAM MODEL

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

$$PP = \exp \left(-\frac{1}{N} \sum_{i=1}^N \log \left(\frac{\text{count}(w_i, w_{i-1}, w_{i-2})}{\text{count}(w_{i-1}, w_{i-2})} \right) \right) \quad (34)$$

where

PP is the perplexity of the language model

N is the number of words in the test set

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:

$$PP = \exp \left(-\frac{1}{N} \sum_{i=1}^N \log \left(\frac{\text{count}(w_i, w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})}{\text{count}(w_{i-1}, w_{i-2}, \dots, w_{i-(n-1)})} \right) \right) \quad (35)$$

where PP is the perplexity of the language model

PERPLEXITY - EXAMPLE I

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:

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

$$\begin{aligned} PP &= \sqrt[3]{\frac{1}{P(I \text{ like cats})}} \\ &= \sqrt[3]{\frac{1}{0.003125}} = 6.84 \end{aligned}$$

The models uncertainty is equivalent to choosing uniformly among about 7 possible next words on average, as indicated by the $PP = 6.84$ value.

UNSEEN SENTENCE/WORD EXAMPLE I

Lets calculate the perplexity for the unseen sentence I like rabbits. Smoothing can be applied to rabbits since its unseen in the training corpus. We can either smooth all bigrams or only for unseen words. In this case, were smoothing only for unseen tokens/words.

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

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

UNSEEN SENTENCE/WORD EXAMPLE II

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

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

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

UNSEEN SENTENCE/WORD EXAMPLE III

The perplexity of this unseen sentence is

$$\begin{aligned}\text{Perplexity}_{\text{unseen}} &= \sqrt[3]{\frac{1}{P(\text{I like rabbits})}} \\ &= \sqrt[3]{\frac{1}{1.25e-07}} \\ &= 200\end{aligned}$$

The perplexity for the seen sentence "I like cats" is approximately 6.84. However, the perplexity for the unseen sentence "I like rabbits" is much higher (200) as it contains an unseen word, *rabbits*.

Perplexity serves as a metric to evaluate language models, where lower values indicate better model performance in predicting given sentences.

CURSE OF DIMENSIONALITY

Vocabulary size = V . The parameters for

- ▶ The curse of dimensionality refers to the exponential increase in the size of the parameter space as the dimensionality of the data increases.

- ▶ It becomes apparent in the context of an n -gram language model due to the rapidly growing number of parameters as the value of n increases.

CHALLENGES

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** - n-gram models may become overly sensitive to specific patterns in the training data and may not generalize well
- ▶ **Computational Complexity** - Training and inference become computationally expensive as the number of parameters grows

TECHNIQUES

- ▶ **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

BACK-OFF AND INTERPOLATION

- ▶ Higher and lower order n-gram models exhibit distinct strengths and weaknesses.
- ▶ Higher order n-grams are sensitive to a broader context, but their counts are sparse.
- ▶ Lower order n-grams consider only a limited context, but their counts are robust.

$$p(w_3|w_1, w_2) = \lambda_1 p_1(w_3) + \lambda_2 p_2(w_3|w_2) + \lambda_3 p_3(w_3|w_1, w_2) \quad (36)$$

The lambdas ($\lambda_1, \lambda_2, \lambda_3$) are weights that determine the contribution of each n-gram order to the final probability estimate. They control how much we trust each level of context.

The key constraint on the lambdas:

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \quad (37)$$

EXERCISE

- ▶ Extend the program^{4,5} to compute the probability and perplexity of a test sentence
 1. Build the language model using corpus
 - ▶ Big corpus → Long time to build a model
 - ▶ Development → Choose a smaller corpus
 - ▶ Testing/Production → use a bigger corpus to build your model
 2. Check the model parameters using the debugger
 3. Once satisfied with the learned model, test your sentences using the model
- ▶ **Input:**
 1. A sentence consisting of words in the corpus that you have used for creating the language model
 2. A sentence with one more words that are OOV
- ▶ **Output:**Probability and perplexity of the input sentence
- ▶ Try this exercise for trigram and 4-gram language models

⁴<https://github.com/Ramaseshanr/anlp/blob/master/BigramLM.ipynb>

⁵<https://github.com/Ramaseshanr/anlp/blob/master/TrigramLM.ipynb>