Language Modeling

Language models predict the probability of upcoming words in a sequence. They can assign probabilities to:

- The next word in a sentence
- A whole sentence or phrase

Types of Language Models

1. **Statistical Language Models**: These rely on traditional statistical techniques, including N-grams, Hidden Markov Models (HMM), and linguistic rules.
2. **Neural Language Models**: These use neural networks to predict word sequences, often achieving higher accuracy.

Probabilistic Language Models

The goal of probabilistic models is to compute the likelihood of word sequences. This allows for:

- Better machine translation (e.g., "high winds tonight" is more likely than "large winds tonight")
- Spell correction (e.g., "minutes" is more likely than "minuets")

Mathematical Representation

- The probability of a sequence of words $P(W)$ is computed as:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

- The probability of the next word, given the previous words, is:

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

- The **Chain Rule of Probability** helps break this down further to estimate each word's probability.

N-Gram Language Models:

Introduction to N-Grams

An **N-gram** is a contiguous sequence of 'N' items (words or tokens) from a given sequence of text or speech. N-grams are used in **language modeling**, which is essential in **natural language processing (NLP)** tasks, such as speech recognition, machine translation, and predictive text generation.

The idea behind N-grams is that the probability of a word depends on the previous words, and N-grams model these dependencies. Here's how N-grams work at various levels:

- **1-gram (Unigram)**: Single words (independent assumption).
- **2-gram (Bigram)**: Sequences of two consecutive words.

- **3-gram (Trigram)**: Sequences of three consecutive words.

N-Gram Models

1. Unigram Model

The **unigram model** assumes that each word is independent of the others. The probability of a sequence of words is simply the product of the probabilities of each word in the sequence:

$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1) \cdot P(w_2) \cdot P(w_3) \cdot \cdots \cdot P(w_n)$$

Example:
For the sentence "I love data science," the unigram model would calculate the probability as:

$$P(\text{"}I\text{"}) \times P(\text{"}love\text{"}) \times P(\text{"}data\text{"}) \times P(\text{"}science\text{"})$$

2. Bigram Model

In the **bigram model**, we consider pairs of consecutive words. The probability of a word depends on the previous word. The probability of a sequence of words is the product of conditional probabilities:

$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \cdots \cdot P(w_n|w_{n-1})$$

Example:
For the sentence "I love data science," the bigram model would calculate the probability as:

$$P(\text{"}I\text{"}) \times P(\text{"}love\text{"}|\text{"}I\text{"}) \times P(\text{"}data\text{"}|\text{"}love\text{"}) \times P(\text{"}science\text{"}|\text{"}data\text{"})$$

3. Trigram Model

In the **trigram model**, the probability of a word depends on the previous two words. This is useful for modeling short context-dependent phrases.

$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot \cdots \cdot P(w_n|w_{n-2}, w_{n-1})$$

Example:
For the sentence "I love data science," the trigram model would calculate the probability as:

$$P(\text{"}I\text{"}) \times P(\text{"}love\text{"}|\text{"}I\text{"}) \times P(\text{"}data\text{"}|\text{"}Ilove\text{"}) \times P(\text{"}science\text{"}|\text{"}lovedata\text{"})$$

How N-Gram Models Work

Step 1: Define N-grams

For a sentence, you first need to define your N-grams. Let's consider the sentence:

Sentence: "I love reading blogs about data science."

Here are the possible N-grams:

- **1-grams (Unigrams)**: "I", "love", "reading", "blogs", "about", "data", "science."
- **2-grams (Bigrams)**: "I love", "love reading", "reading blogs", "blogs about", "about data", "data science."
- **3-grams (Trigrams)**: "I love reading", "love reading blogs", "reading blogs about", "blogs about data", "about data science."

Step 2: Estimate Probabilities

The next step is to estimate the probabilities for each N-gram using the Chain Rule. You do this by counting how often an N-gram appears in the corpus and then normalizing it with the total number of N-grams.

$$\mathbf{p(B|A) = P(A,B)/P(A)} \quad \text{Rewriting:} \quad \mathbf{P(A,B) = P(A)P(B|A)}$$

Example:
For the bigram "I love":

- **Count** of "I love" = 10
- **Count** of "I" = 100
- **Probability**:

$$P(\text{"}love\text{"}|\text{"}I\text{"}) = \frac{Count(\text{"}Ilove\text{"})}{Count(\text{"}I\text{"})} = \frac{10}{100} = 0.1$$

Step 3: Compute Sentence Probability

Once you have the probabilities of N-grams, you can calculate the probability of an entire sentence. This is done by multiplying the probabilities of the individual N-grams.

Example:
For the bigram model, the sentence "I love data science" has the probability:

$$P(\text{"}Ilovedatascience\text{"}) = P(\text{"}I\text{"}) \times P(\text{"}love\text{"}|\text{"}I\text{"}) \times P(\text{"}data\text{"}|\text{"}love\text{"}) \times P(\text{"}science\text{"}|\text{"}data\text{"})$$

If the probabilities are:

- $P(\text{"}I\text{"}) = 0.05$
- $P(\text{"}love\text{"}|\text{"}I\text{"}) = 0.3$
- $P(\text{"}data\text{"}|\text{"}love\text{"}) = 0.2$
- $P(\text{"}science\text{"}|\text{"}data\text{"}) = 0.15$

Then the sentence probability is:

$$P(\text{"}Ilovedatascience\text{"}) = 0.05 \times 0.3 \times 0.2 \times 0.15 = 0.00045$$

MORE EXAMPLES: BERKELEY RESTAURANT PROJECT SENTENCES

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

RAW BIGRAM COUNTS

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

RAW BIGRAM PROBABILITIES

- Normalize by unigrams:

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

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

BIGRAM ESTIMATES OF SENTENCE PROBABILITIES

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

Smoothing Techniques

One common issue with N-gram models is **data sparsity**—many possible word sequences are not observed in the training data. To handle this, we use **smoothing** techniques, which assign small probabilities to unseen N-grams:

Add-One (Laplace) Smoothing:
Add 1 to all counts to ensure no zero probabilities. For example:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + V}$$

where V is the total number of unique words (vocabulary size).

LAPLACE-SMOOTHED BIGRAMS

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

RECONSTITUTED COUNTS

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

COMPARE WITH RAW BIGRAM COUNTS

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

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Perplexity: Evaluating N-Gram Models

Perplexity is a metric used to evaluate the quality of a language model. It measures how well a model predicts a sample of data. A **lower perplexity** indicates a better model.

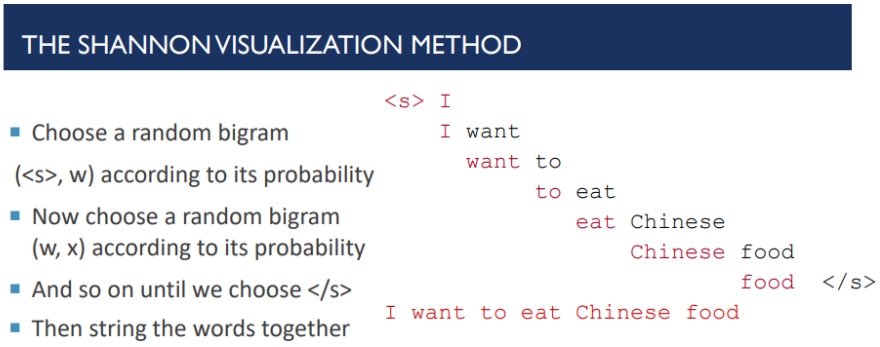
The formula for perplexity is:

$$PP(W) = P(w_1, w_2, \ldots, w_N)^{-1/N}$$

Where:

- W is the sequence of words,
- N is the total number of words.

In practice, we compute the perplexity using **log probabilities** to avoid underflow.



Exercise.pdf

Q1. We are given the following corpus:

<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like that Sam </s>

Bigrams: <s>, I = 3
I, am = 3
am, Sam = 2
Sam, </s> = 3
<s>, Sam = 1
Sam I = 1
am, </s> = 1
I, do = 1
do, not = 1
not, like = 1
like, that = 1
that, Sam = 1

$V = 9$

$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$

Using a bigram language model with add-one smoothing, create a term matrix. Include <s> and </s> in your counts just like any other token.

	<s>	I	am	Sam	do	not	like	that	</s>
<s>	0.0769	0.3076	0.0769	0.1538	0.0769	0.0769	0.0769	0.0769	0.0769
I	0.0769	0.0769	0.3076	0.0769	0.1538	0.0769	0.0769	0.0769	0.0769
am	0.0833	0.0833	0.0833	0.25	0.0833	0.0833	0.0833	0.0833	0.166
Sam	0.0769	0.1538	0.0769	0.0769	0.0769	0.0769	0.0769	0.0769	0.3076
do	0.1	0.1	0.1	0.1	0.1	0.2	0.1	0.1	0.1
not	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1	0.1
like	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1
that	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.1
</s>	0.0769	0.0769	0.0769	0.0769	0.0769	0.0769	0.0769	0.0769	0.0769