Natural Language Processing with Probabilistic Models

Zeham Management Technologies BootCamp by SDAIA

September 4th, 2024

Agenda

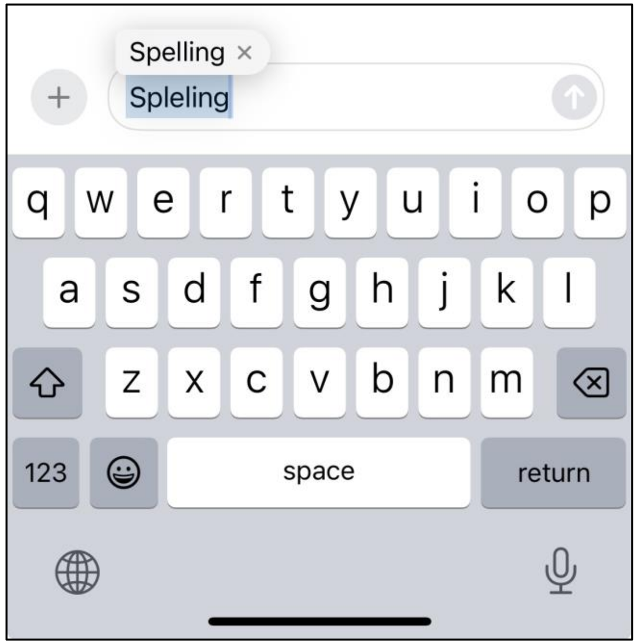
- Auto-correct
- Auto-complete
- Word Embeddings with Neural Networks

Auto-correct

What is Auto-correct?

Auto-correct is a feature you use every day. Whether you're sending a text message to a friend or making a typo in a search query, auto-correct works behind the scenes to correct your sentences.

Auto-correct is a feature that corrects misspelled words by automatically changing them to their correct forms.



How Auto-Correct Works?

1. Identify the misspelled word.
2. Find strings that are n edits away from the given word.
3. Filter Candidates.
4. Compute word probabilities.

Identify the misspelled word

Identify the misspelled word

To identify a misspelled word, you first need a vocabulary of correctly spelled words to compare against.

How to Build a Vocabulary:

The vocabulary can be built by using text files that store all correct words and store it in a set.

Identify the misspelled word

Tuwaiq Academy is the first academy in the Kingdom of Saudi Arabia to offer educational programs in advanced technologies, with a wide range of courses for different age groups and technical fields. It was founded in August 2019

↓

Vocab
Tuwaiq
Academy
Is
…
August

Identify the misspelled word

Words not found in the vocabulary are considered misspelled.

Example:

Tuwaiq Academy is teh first academy in the Kingdom of Saudi Arabia.

Misspelled Word:

"teh"

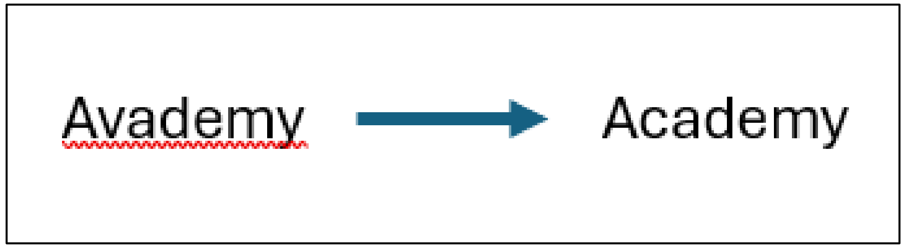
Vocab
Tuwaiq
Academy
Is
…
August

Find strings that are n edits away from the given word

n edit distance

n Edits: The number of character modifications needed to transform the observed string into a target string.

Observed string: The typo.
Target string: The potential candidate for the suggestion (Correct string).



n edit distance

Example for 1 edit distance case:

$$\text{Vat} \longrightarrow \text{Cat}$$

Example for 2 edit distance case:

$$\text{Speed} \longrightarrow \text{Greed}$$

Types of Edit Manipulations

1. Insert
2. Delete
3. Switch
4. Replace

Types of Edit Manipulations

Insert:

The insert operation involves adding one letter to a word.

eed $\longrightarrow$ need, seed, feed

Delete:

The delete operation involves removing one letter from a word without replacement.

row $\longrightarrow$ ow, rw, ro

Types of Edit Manipulations

Switch:
The switch operation involves swapping two letters in a word.

$$\text{eat} \longrightarrow \text{eta, ate}$$

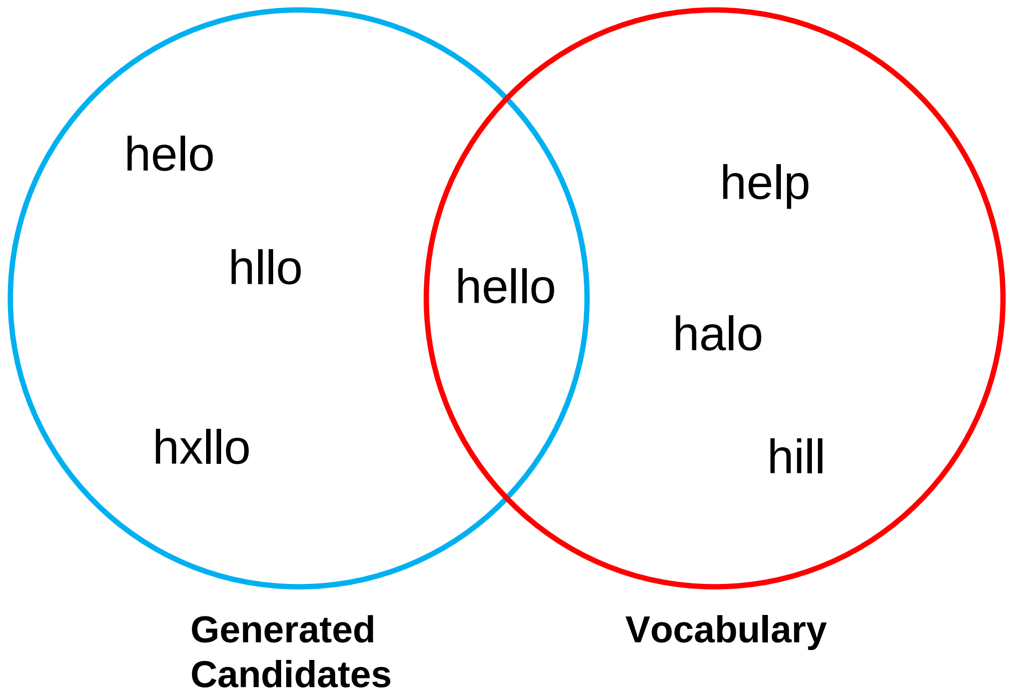
Replace:
The replace operation involves replacing one letter with another in a word.

$$\text{car} \longrightarrow \text{cat, cap}$$

Filter Candidates

Filter Candidates

In this step we want to ensure that all strings generated in step 2 are meaningful words that exist in the vocabulary.



Compute word probabilities

Compute word probabilities

Here's where we can leverage the power of probabilistic models in NLP.

Our goal is to select the candidate with the highest probability of appearing in the given context.

Formula:

$$P(w) = \frac{C(w)}{V}$$

Where:
$P(w)$ is the probability of a word.
$C(w)$ is the number of times the word appears.
V is the total size of the corpus.

Compute word probabilities

Example:
“I am happy because I am in Tuwaiq”.

We will compute the probability of the word “am”.

$$P(am) = \frac{C(am)}{V} = \frac{2}{8}$$

Word	Count
I	2
am	2
happy	1
because	1
In	1
Tuwaiq	1
Total: 8	

Let's Practice

Tutorial:

- 7- Introduction to Natural Language Processing/3 - Natural Language Processing with Probabilistic Models/LAB/Auto_Correction.ipynb

Exercise:

- 7- Introduction to Natural Language Processing/3 - Natural Language Processing with Probabilistic Models/LAB/Auto_Correction_exercise.ipynb

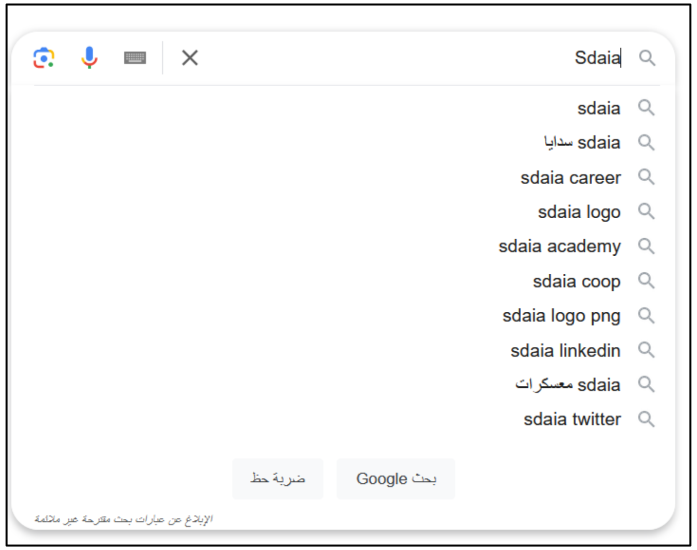
Auto-complete

What is Auto-complete?

Auto-complete is a feature that predicts the next word or phrase as you type.

Auto-complete helps users type faster and with fewer errors by offering suggestions.

Common Uses: Found in search engines, text editors, and messaging apps to improve user experience.



Sdaia

sdaia
sdaia سدايا
sdaia career
sdaia logo
sdaia academy
sdaia coop
sdaia logo png
sdaia linkedin
sdaia معسكرات
sdaia twitter

Understanding N-grams

What are N-grams?

Definition:
N-grams are sequences of 'n' consecutive items, typically words in the context of NLP. The value of 'n' determines the complexity and the amount of context captured by the N-gram.

Why Should We Know N-grams?
N-grams are essential for building language models used in tasks like auto-complete, where they predict the next word based on the preceding words in a sequence.

What are N-grams?

Corpus: Sdaia T5 is the best bootcamp

Uni-gram (n=1)	Sdaia	T5	Is	The	Best	Bootcamp

Bi-gram (n=2)	Sdaia T5	T5 is	Is the	The best	Best bootcamp

Tri-gram (n=3)	Sdaia T5 is	T5 is the	Is the best	The best bootcamp

How N-grams Work in Auto-Complete?

Sequence Notation

Corpus: Tuwaiq academy is a great place … artificial intelligence and data science

$w1$ $w2$ $w3$ $w4$ $w5$ $w6$ $w296$ $w297$ $w298$ $w299$ $w300$

$$m = 300$$

$$w_1^m = w_1\, w_2 \;\; \ldots \;\; w_m$$

$$w_1^4 = w_1\, w_2\, w_3\, w_4$$

Unigram Probability

Corpus: I am happy because I am in Tuwaiq

Size of corpus m = 8

$$P\ (I) = \frac{2}{8} = \frac{1}{4}$$

$$P\ (Tuwaiq) = \frac{1}{8}$$

Probability of unigram: $$P\ (w) = \frac{C(w)}{m}$$

Bigram Probability

Corpus: I am happy because I am in Tuwaiq

Probability of “I am” : $$P\ (am|I) = \frac{C(I\ am)}{C(I)} = \frac{2}{2} = 1$$

Probability of “I Tuwaiq”: $$P\ (Tuwaiq|I) = \frac{C(I\ Tuwaiq)}{C(I)} = \frac{0}{2} = 0$$

Probability of “am happy”: $$P\ (happy|am) = \frac{C(am\ happy)}{C(am)} = \frac{1}{2}$$

Bigram Probability

Corpus: I am happy because I am in Tuwaiq

Probability of bigram: $$P\,(y|x) = \frac{C(x\,y)}{\sum_w C(x\,w)} = \frac{C(x\,y)}{C(x)}$$

Trigram Probability

Corpus: I am happy because I am in Tuwaiq

Probability of "I am happy":

$$P\ (happy|I\ am) = \frac{C(I\ am\ happy)}{C(I\ am)} = \frac{1}{2}$$

Probability of trigram:

$$P(w3|w_1^2) = \frac{\boxed{C(w_1^2\ w_3)}}{C(w_1^2)} = \frac{C(w_1^3)}{C(w_1^2)}$$

$$C(w_1^2\ w_3) = \mathrm{C}(\ w_1\ w_2\ w_3) = C(w_1^3)$$

N-gram Probability

Probability of bigram: $P\,(w_N|w_1^{N-1}) = \frac{C(w_1^{N-1}\ w_N)}{C(w_1^{N-1})} = \frac{C(w_1^{N})}{C(w_1^{N-1})}$

Probability of a Sequence

Probability of a sequence

What is the probability of the following sentence?

$$P(My\ friend\ drinks\ coffee) = ?$$

Conditional probability and chain rule:

$$P(A, B, C, D) = P(A)\ P(B|A)\ P(C|A, B) P(D|A, B, C)$$

Probability of a sequence

What is the probability of the following sentence?

$$P(My\ friend\ drinks\ coffee)$$

$$= P(My)\ P(friend|My)\ P(drinks|My\ friend) \boxed{P(coffee|My\ friend\ drinks)}$$

$$P(coffee|My\ friend\ drinks) = \frac{C(My\ friend\ drinks\ coffee)}{C(My\ friend\ drinks)}$$

It seems there is a problem with this method! 🤔

Probability of a sequence

Challenge: Natural language is extremely variable, making it unlikely that entire sentences or even longer phrases will appear exactly as they are in the training data.

Using our previous example, the formula looks for the count of the full phrase “My friend drinks coffee" and its prefix “My friend drinks“.

Issue: It’s unlikely that both the phrase and its prefix will exist in the training data in the exact order. As a result, **their counts are ZERO.**

Impact: With zero counts, the formula can't provide a valid probability estimate for the entire sentence.

Probability of a sequence

The Markov Assumption:
The Markov assumption simplifies the prediction by assuming that the probability of a word depends only on the previous word, not the entire sentence.

We will use a Bigram model (n=2), so a first order Markov assumption will be made.

$$P(My)\ P(friend|My)\ P(drinks|My\ friend) P(coffee|My\ friend\ drinks)$$

$$\Downarrow$$

$$P(My)\ P(friend|My)\ P(drinks|friend) P(coffee|drinks)$$

Modeling the Entire Sentence Using Bigrams: $P(w_1^n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$

Start of Sentence

In a bigram model, the first word of a sentence has no preceding word to form a pair, making it difficult to calculate its probability directly.

My friend drinks coffee

⬇

$$\boxed{P(My)}\ P(friend|My)\ P(drinks|friend)P(coffee|drinks)$$

In this case we will use **<s>** to act as a prefix for the first word in the sentence.

<s> My friend drinks coffee

⬇

$$P(<s>|My)\ P(friend|My)\ P(drinks|friend)P(coffee|drinks)$$

End of Sentence

Similarly, we add the **</s>** marker at the end of the sentence to indicate its conclusion.

<s> My friend drinks coffee </s>

⬇

$$P(<s>|My)\ P(friend|My)\ P(drinks|friend)P(coffee|drinks)\ \mathrm{P}(</s>|\mathrm{coffee})$$

This helps the model understand that the sentence is complete and prepares it for the next input.

Count Matrix & Probability Matrix

Count Matrix

What is a Count Matrix?
A count matrix is a table that records the frequency of word pairs (bigrams) in the training data.

Each cell in the matrix represents the number of times a specific bigram appears in the corpus.

Count Matrix

$$P(w_n | w_{n-N+1}^{n-1}) = \frac{\boxed{C(w_{n-N+1}^{n-1}, w_n)}}{C(w_{n-N+1}^{n-1})}$$

Bigram count matrix:

Corpus: <s> I learn I grow </s>

		<s>	</s>	I	learn	grow
<s>I	<s>	0	0	1	0	0
	</s>	0	0	0	0	0
I learn – I grow	I	0	0	0	1	1
Learn I	learn	0	0	1	0	0
grow I	grow	0	1	0	0	0

Probability Matrix

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{\boxed{C(w_{n-N+1}^{n-1})}}$$

The probability matrix is derived directly from the count matrix by normalizing the counts (dividing each count by the total occurrences of the first word).

Corpus: <s> I learn I grow </s>

Count Matrix

	<s>	</s>	I	learn	grow	Sum
<s>	0	0	1	0	0	1
</s>	0	0	0	0	0	0
I	0	0	0	1	1	2
learn	0	0	1	0	0	1
grow	0	1	0	0	0	1

$\longrightarrow$

Probability Matrix

	<s>	</s>	I	learn	grow
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	0.5	0.5
learn	0	0	1	0	0
grow	0	1	0	0	0

Language Model

Probability matrix is used for:

- Sentence probability
- Next word prediction

Probability Matrix

	<s>	</s>	I	learn	grow
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	0.5	0.5
learn	0	0	1	0	0
grow	0	1	0	0	0

Sentence probability:

<s> I grow </s>

$$P(< s > I\ grow < / s >)$$

$$= P(I| < S >)\ P(grow|I)\ P(< / s > |grow)$$

$$= 1\ \times 0.5 \times 1$$

$$= 0.5$$

Language Model Evaluation

Perplexity

What is Perplexity?
a metric used to evaluate how well a language model predicts a sample of text.

It measures the uncertainty or "confusion" of the model when predicting the next word in a sequence.

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Perplexity

Perplexity for Bigrams:

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

✅**Low Perplexity:** Indicates the model predicts the sequence with high confidence.

❌**High Perplexity:** Suggests that the model is uncertain, and the predictions are less accurate.

Improving N-grams with Smoothing Techniques

Smoothing

Smoothing: Redistributing a small portion of the probability mass from more frequent events to account for unseen events.

Bigrams	Frequency
Tuwaiq academy	6
Tuwaiq is	3
Tuwaiq in	2
Tuwaiq helps	0

Bigrams	Frequency
Tuwaiq academy	5
Tuwaiq is	3
Tuwaiq in	2
Tuwaiq helps	1

Smoothing

Some Smoothing Techniques:

- Laplace (add-one)
- Add-k
- Backoff: Katz Backoff

Next, we'll dive into each technique.

Laplace (add-one)

- Add one to each n-gram count before converting them into probabilities.
- Not the most effective technique for language modeling, but it serves as a useful baseline.

$$P(w_i) = \frac{C(w_i)}{N} \quad \Rightarrow \quad P_{Laplace}(w_i) = \frac{C(w_i) + 1}{N + V}$$

Laplace (add-one)

Unigram	Frequency
Tuwaiq	4
is	8
amazing	6
helpful	0

Bigram	Frequency
Tuwaiq is	2
is amazing	4
is helpful	0

$$P(w_i) = \frac{C(w_i)}{N}$$

Unigram	Probability
Tuwaiq	$\frac{4}{18} = 0.22$
is	$\frac{8}{18} = 0.44$
amazing	$\frac{6}{18} = 0.33$
helpful	$\frac{0}{18} = 0.00$

Bigram	Probability
Tuwaiq is	$\frac{2}{4} = 0.50$
is amazing	$\frac{4}{8} = 0.22$
is helpful	$\frac{0}{8} = 0.00$

Laplace (add-one)

Now let's apply Laplace technique.

$$P(w_i) = \frac{C(w_i)}{N} \Rightarrow P_{Laplace}(w_i) = \frac{C(w_i)+1}{N+V}$$

Unigram	Frequency
Tuwaiq	4+1
is	8+1
amazing	6+1
helpful	0+1

Bigram	Frequency
Tuwaiq is	2+1
is amazing	4+1
is helpful	0+1

$$P_{Laplace}(w_i) = \frac{C(w_i)+1}{N+V} \Rightarrow$$

V is the size of the vocabulary (V=4)

Unigram	Probability
Tuwaiq	$\frac{5}{22} = 0.23$
is	$\frac{9}{22} = 0.41$
amazing	$\frac{7}{22} = 0.32$
helpful	$\frac{1}{22} = 0.05$

Bigram	Probability
Tuwaiq is	$\frac{3}{4+4} = \frac{3}{8} = 0.38$
is amazing	$\frac{5}{8+4} = \frac{5}{12} = 0.42$
is helpful	$\frac{1}{8+4} = \frac{1}{12} = 0.08$

Laplace (add-one)

Before Laplace technique

Bigram	Probability
Tuwaiq is	$\frac{2}{4} = 0.50$
is amazing	$\frac{4}{8} = 0.22$
is helpful	$\frac{0}{8} = 0.00$

After Laplace technique

Bigram	Probability
Tuwaiq is	$\frac{3}{4+4} = \frac{3}{8} = 0.38$
is amazing	$\frac{5}{8+4} = \frac{5}{12} = 0.42$
is helpful	$\frac{1}{8+4} = \frac{1}{12} = 0.08$

Add-K Smoothing

- Moves a bit less of the probability mass from seen to unseen events.
- Instead of adding one to each count, a fractional count is added.
  - For example: 0.5, 0.06, or 0.01.
- The value of kk can be fine-tuned using a validation set.

$$P(w_i) = \frac{C(w_i)}{N} \quad \Longrightarrow \quad P_{Add-K}(w_i) = \frac{C(w_i) + k}{N + kV}$$

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

Backoff

Backoff is a strategy used when an n-gram has zero counts.

How It Works:

- If the required n-gram has zero occurrences, the model "backs off" to the (n-1)-gram.
- This process continues until a gram size with non-zero counts is found.

probability mass from higher-order n-grams is redistributed to lower-order n-grams to ensure accurate predictions.

Katz Backoff

Katz Backoff refines the basic backoff approach by using discounted probabilities for seen n-grams and redistributing mass to lower-order n-grams for unseen ones.

Katz backoff utilizes a **function** $\boldsymbol{\alpha}$ to reallocate probability mass from higher-order n-grams to lower-order n-grams.

Katz Backoff

Discounted Probability P^* :

When an n-gram has non-zero counts, Katz backoff relies on a discounted probability P^* to estimate the likelihood.

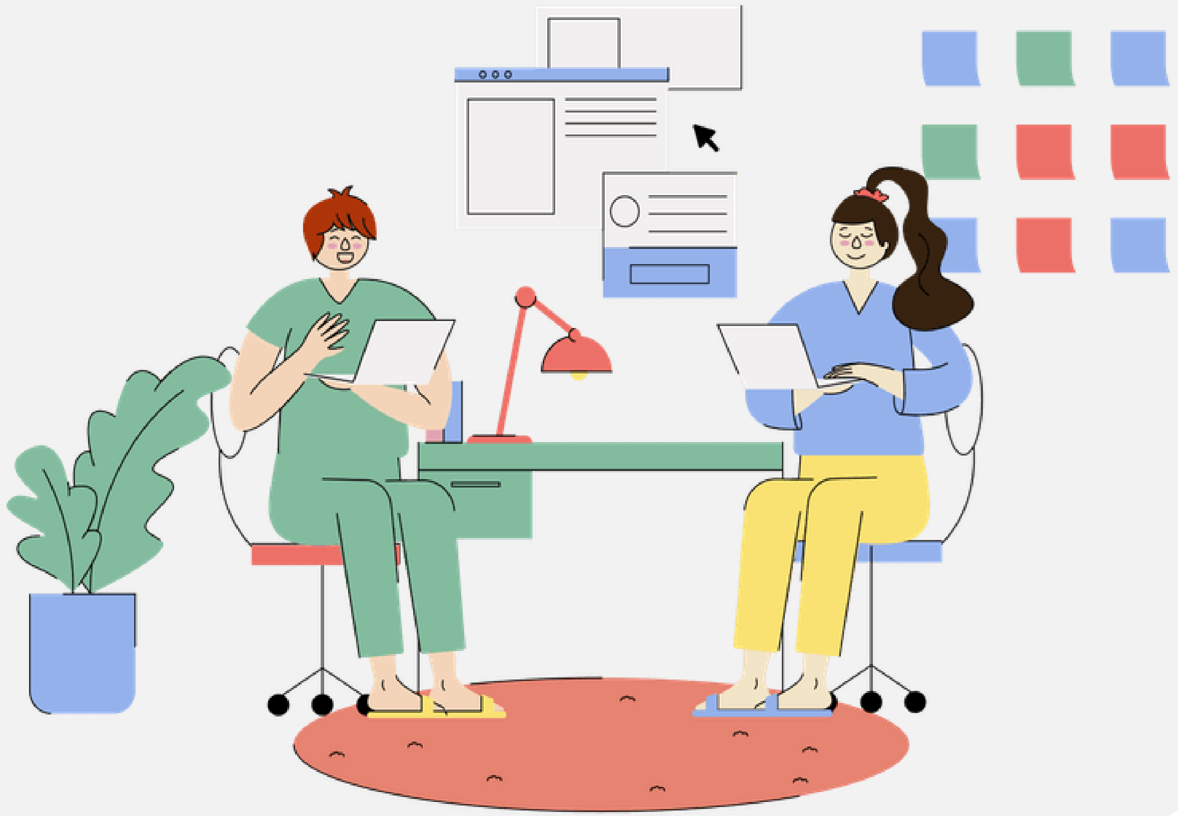
But if the required n-gram count is zero, the model recursively backs off to the Katz probability of the lower-order (n-1)-gram.

$$P_{BO}(w_n | w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n | w_{n-N+1}^{n-1}), & if\ c(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})\, P_{BO}(w_n | w_{n-N+2}^{n-1}), & otherwise \end{cases}$$

Let's Practice

Tutorial:

- 7- Introduction to Natural Language Processing/3 - Natural Language Processing with Probabilistic Models/LAB/Auto_Complete_Using_N_Grams.ipynb



Word Embeddings with Neural Networks

Why Neural Networks for Word Embeddings?

Learning from Context: Neural networks excel at capturing the context in which words appear, allowing them to learn complex relationships between words based on their surrounding words in a sentence.

Dimensionality Reduction: Traditional representations like one-hot encoding create sparse, high-dimensional vectors. Neural networks, on the other hand, can learn dense, low-dimensional embeddings that retain meaningful information while being computationally efficient.

Why Neural Networks for Word Embeddings?

Semantic Relationships: Neural networks are capable of learning embeddings where words with similar meanings are placed closer together in the vector space. This enables the model to capture subtle nuances in language, such as synonyms or related concepts.

Training on Large Corpora: Neural networks can be trained on vast amounts of text data, allowing them to learn word embeddings that are robust and capture the diverse ways in which words are used across different contexts and domains.

Word2Vec

Word2Vec

What is Word2Vec?

- Word2Vec is a neural network-based technique for generating word embeddings.
- It represents words as continuous vectors in a high-dimensional space, where similar words are mapped to vectors that are close together.

The goal of Word2Vec is to capture the semantic relationships between words, allowing words with similar meanings to have similar vector representations.

Word2Vec is developed by Google, it has become a widely-used method for creating word embeddings due to its efficiency and effectiveness.

Word2Vec

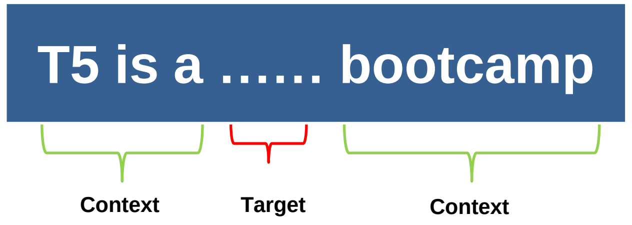
Word2Vec introduces two model architectures:

- Continuous bag-of-words (CBOW).
- Skip-Gram

CBOW

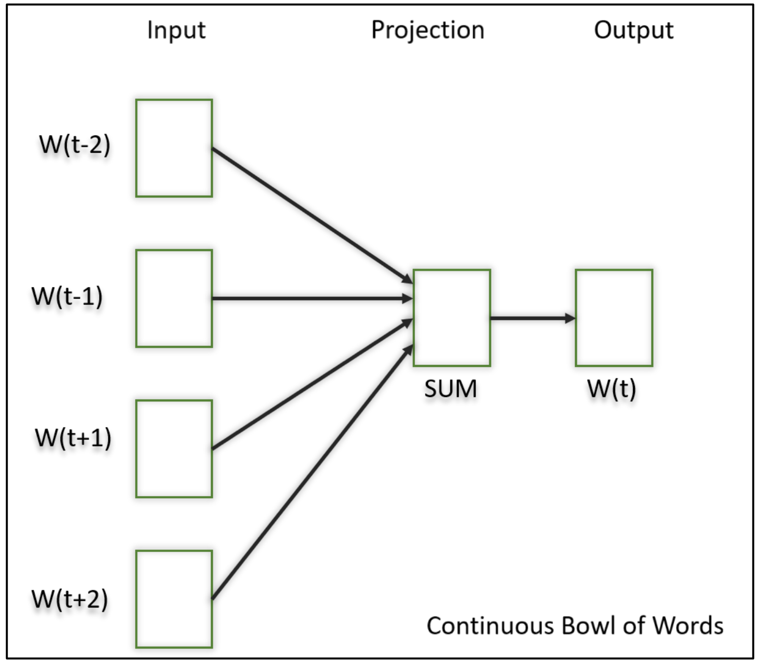
CBOW

CBOW is a neural network algorithm designed to predict a target word based on its surrounding context words.



Architecture of the CBOW Model

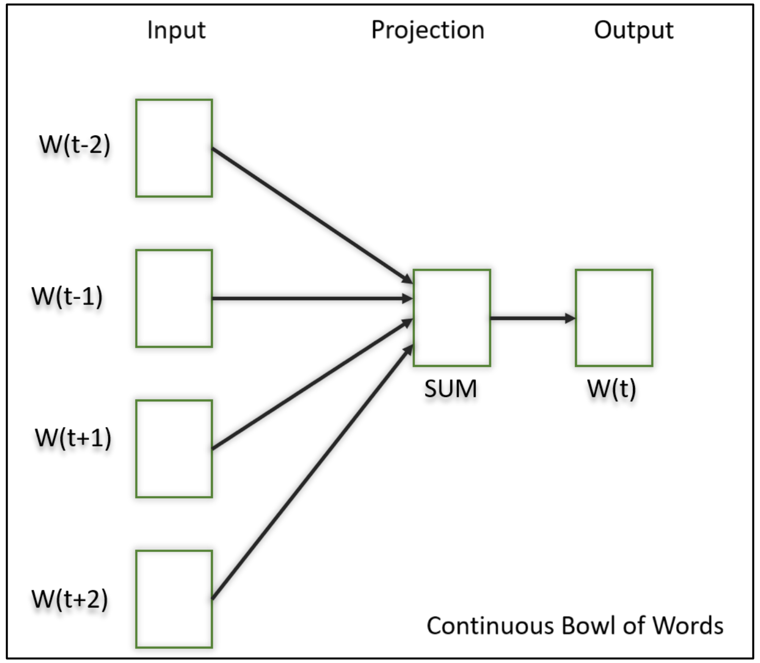
CBOW is a feedforward neural network with a single hidden layer. The input layer consists of the context words, while the output layer predicts the target word.



Source

Architecture of the CBOW Model

The hidden layer contains the continuous vector representations (word embeddings) learned from the input words.



Source

Advantages of CBOW

Faster Training: CBOW is quicker to train than Skip-gram, particularly on large datasets.

Effective with Frequent Words: It excels in scenarios where context is more important than the position of individual words, making it ideal for capturing the meanings of common words.

Disadvantages of CBOW

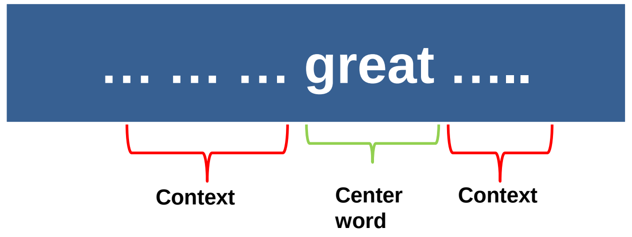
Struggles with Rare Words: Less effective at capturing rare words or phrases.

No Word Order: Does not preserve the order of words, which can be important in some applications.

Skip-Gram

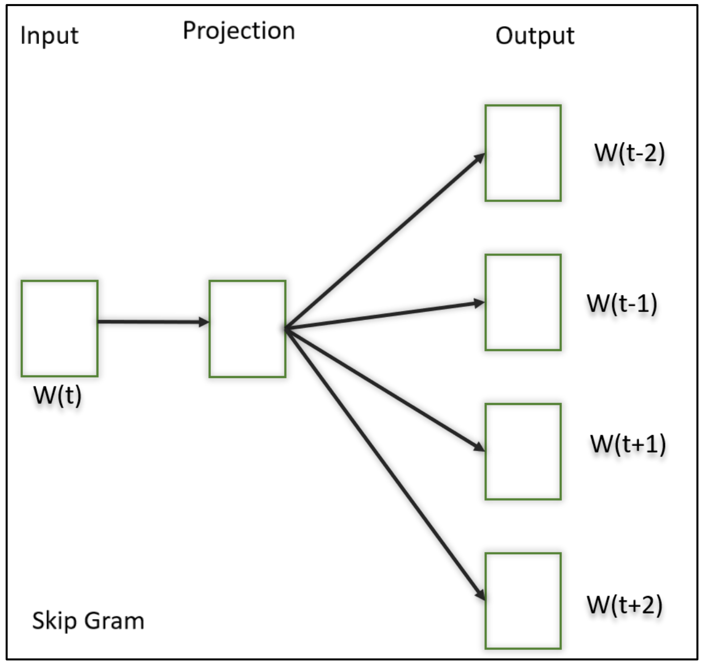
Skip-Gram

In the Skip-Gram model, the input is the center word, and the model predicts the surrounding context words.



Architecture of the Skip-Gram Model

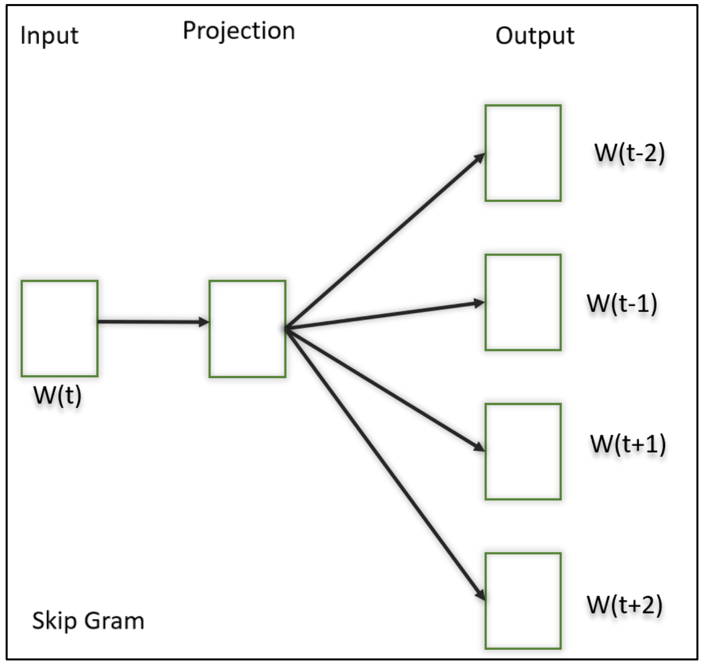
Input Layer: Takes a single word (the target word) and represents it as a one-hot vector.



Source

Architecture of the Skip-Gram Model

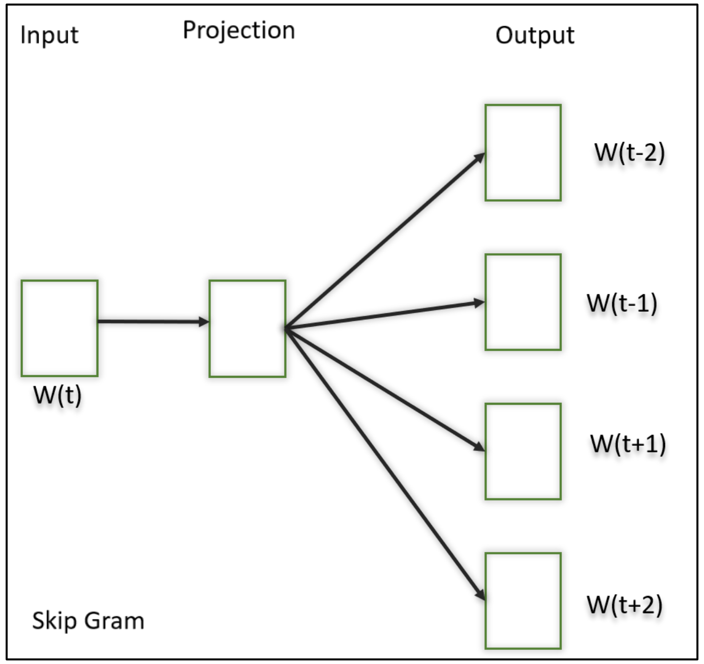
Hidden Layer: Transforms the input word into a continuous, distributed representation within the hidden layer.



Source

Architecture of the Skip-Gram Model

Output Layer: Predicts the context words (surrounding words) using the learned representation from the hidden layer.



Source

Advantages of Skip-Gram

Effective with Rare Words: Excels at capturing low-frequency words by predicting multiple context words.

Preserves Word Order: Maintains the sequence of words, important for tasks like translation.

Disadvantages of Skip-Gram

Computationally Heavy: More resource-intensive to train, especially with large datasets.

Less Effective with Frequent Words: May not perform as well with very common words compared to CBOW.

Let's Practice

Tutorial:

- 7- Introduction to Natural Language Processing/3 - Natural Language Processing with Probabilistic Models/LAB/ CBOW_Word_Embeddings.ipynb

Thank You

SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority