

# NLP

## 1.Bag of Words (BoW)

* The text is represented as a sparse vector of word counts or binary values.
* It doesn’t consider word order or semantics.

#### Disadvantages :

* High-dimensional for large vocabularies (sparse representation).
* Struggles with contextual understanding.

#### When to Use

* Structured Text Data: Works well with small or moderately sized datasets with predictable and structured text (e.g., reviews, simple classification tasks).
* Example: Spam detection, sentiment analysis.
* Text with Simple Patterns: When the frequency of words matters more than their order or context.
* Example: Classifying product reviews based on specific keywords (e.g., "good", "bad").

#### When NOT to Use:

* Large and Complex Datasets: BoW struggles with high-dimensional data due to its sparse representation.
* Example: A dataset with millions of unique words (e.g., scientific articles or legal documents).
* Context-Dependent Tasks: When understanding word relationships or context is essential.
* Example: Tasks involving polysemous words like "bank" (river bank vs financial bank).

## 2. GloVe (Global Vectors for Word Representation)

* GloVe creates pre-trained word embeddings based on co-occurrence statistics of words in a corpus. Each word is represented as a dense vector in a fixed-dimensional space.
* Train on a co-occurrence matrix from a large corpus.
* Captures semantic relationships between words (e.g., “king - man + woman = queen”).

#### Disadvantages

* Static embeddings: Each word has a single fixed vector, losing context (e.g., "bank" in different sentences).

In GloVe (Global Vectors for Word Representation), static embeddings refer to the way words are represented by vectors that remain fixed, regardless of the context in which the words appear. This means that each word is assigned a single, unique vector representation that does not change, no matter where or how the word is used in a sentence.

#### How Static Embeddings Work in GloVe

* Word Representation: In GloVe, words are mapped to vectors in a high-dimensional space, and each word has a corresponding vector. These vectors are pre-trained on a large corpus (e.g., Wikipedia, Common Crawl) and capture semantic relationships between words based on their co-occurrence patterns.
* For example, "dog" might have the vector [ 0.25 , − 0.30 , 0.17 , … ] [0.25,−0.30,0.17,…], and "cat" might have the vector [ 0.28 , − 0.32 , 0.14 , … ] [0.28,−0.32,0.14,…].
* Fixed Embeddings: These vectors are static because the embedding for a word like "bank" is always the same, regardless of whether the word is used in the context of a river bank or a financial institution. The vector is learned based on overall word co-occurrence statistics in the training corpus, but it does not take into account the surrounding words or sentence structure.

#### When to Use

* **Pre-trained Embeddings**: GloVe is effective for datasets where you don’t want to train embeddings from scratch.
* Example: Similarity search, clustering words, or classification tasks where semantic relationships are important.
* **Moderately Large Datasets**: Works well when the dataset size aligns with the pre-trained embeddings’ corpus.
* Example: Movie reviews, FAQs

#### When NOT to Use:

* **Domain-Specific Datasets**: Pre-trained GloVe embeddings may not align with niche or specialized domains (e.g., medical or legal text).
* **GloVe (Global Vectors for Word Representation)**, as a **pre-trained model**, learns word embeddings based on the statistical patterns of word co-occurrence from a large corpus like Wikipedia or Common Crawl. While this results in effective general-purpose embeddings, it can face challenges when applied to specialized or domain-specific datasets.
* **Pre-trained GloVe Embeddings**: GloVe embeddings are typically trained on a **general-domain corpus**, like news articles, Wikipedia, or web data. While these embeddings capture general relationships between words, they may not perform well for **specific domains** such as **medical, legal, or technical jargon** that weren't extensively represented in the training data.
* **GloVe**: While useful for general tasks, GloVe embeddings may not perform well on domain-specific datasets (e.g., medical or legal texts) because they don't capture the nuanced meanings of specialized jargon. They are static, so they can't adapt to the specific context in which a word is used in a niche field.
* **Solution**: Fine-tuning pre-trained GloVe on domain-specific data or using domain-specific embeddings like **BioWordVec** or **ClinicalBERT** can help improve performance for tasks involving specialized terminologies.
* **Contextual Tasks**: Fails when a single word has multiple meanings in different contexts.

## 3. Word2Vec:

**Word2Vec** is a model that uses **neural networks** to learn dense (continuous) vector representations of words. These vectors capture semantic relationships, so words with similar meanings or contexts end up having similar vector representations.

There are two main ways Word2Vec generates these vector representations:

1. **CBOW (Continuous Bag of Words)**
2. **Skip-gram**

### **CBOW (Continuous Bag of Words)**

**Goal**: Predict the target word (the word in the center) based on the surrounding words (context words).

**How it works**:

* CBOW tries to **predict a word** using the **context words** around it.
* For example, in the sentence:  
  "The cat sits on the mat."  
  If the target word is **"sits"**, the context words could be **"The", "cat", "on", "the", "mat"**. CBOW will take these context words and use them to predict the target word ("sits").

**Intuition**:

* CBOW looks at the surrounding words and tries to figure out what the **center word** (target) is.
* **Example**: In the context of the sentence "The cat sits on the mat," CBOW might use the words "The", "cat", "on", "the", "mat" to predict the center word **"sits"**.

### **Skip-gram**

**Goal**: Predict the context words based on a given word.

**How it works**:

* Skip-gram works **opposite** to CBOW. Instead of predicting the target word from context, it **takes a word** and tries to **predict the context words** around it.
* For example, in the same sentence:  
  "The cat sits on the mat."  
  If the center word is **"sits"**, Skip-gram will try to predict the context words **"The", "cat", "on", "the", "mat".**

**Intuition:**

* Skip-gram focuses on how well a single word can predict the words that surround it.
* Example: Given the word "sits", Skip-gram tries to predict the surrounding words like "The", "cat", "on", "the", "mat".

### **Key Differences:**

* CBOW: Predicts a target word based on context words (context → target).
* Skip-gram: Predicts context words based on a target word (target → context).
* CBOW is faster and works well for smaller datasets.
* Skip-gram is better for larger datasets, as it works well for rare words and captures more detailed word relationships.

#### When to Use

* Domain-Specific Datasets: When you can train embeddings on a large, domain-specific corpus.
* Example: Training embeddings for e-commerce product reviews, medical terminology, or scientific papers.
* Semantic Understanding: For applications requiring semantic and syntactic relationships between words.
* Example: Word analogy tasks or language modeling.

#### When NOT to Use

* Small Datasets: Word2Vec requires a large corpus to generate meaningful embeddings. Small datasets will lead to poor-quality vectors.
* Example: Small datasets with a limited vocabulary.

#### Word2Vec and Domain-Specific Datasets:

even in a specialized field, it can learn **context-dependent relationships** better than GloVe. For example, **"virus"** would have a vector that changes depending on the surrounding words, helping it adapt to both the medical and IT security contexts if trained on those domains.

## Difference between glove and word2vec:

* **GloVe**: While **GloVe** can be trained on a domain-specific dataset, it **lacks context** and offers **static representations** of words. It may not handle domain-specific tasks as well as **Word2Vec**, especially when dealing with **rare words**, **specialized terminology**, or **context-dependent meanings**.
* **Word2Vec**: Performs better for **domain-specific tasks** because it learns embeddings based on **local context**, making it more adaptable to specialized language, **rare terms**, and the **changing meanings** of words in different contexts.

## 4. BERT (Bidirectional Encoder Representations from Transformers)

* BERT uses the Transformer architecture to generate contextualized embeddings. Each word’s embedding depends on the entire sentence.
* Pre-train a model using Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).
* Fine-tune on specific tasks.
* **Contextual embeddings: Same word has different vectors in different contexts.**

### **When to Use**

* Contextual Understanding: For datasets requiring understanding of word meaning based on surrounding text.
  + Example: Question answering, sentiment analysis, named entity recognition.
* Advanced NLP Tasks: When the task involves complex relationships between words and context.
  + Example: Summarization, translation, text generation.
* Pre-Trained Knowledge: For fine-tuning on tasks with limited data but rich context.
  + Example: Small datasets with specific tasks, leveraging transfer learning.

### **When NOT to Use**

* **Simple or Structured Datasets**: BERT is overkill for simple tasks like word frequency analysis or basic classification.
  + Example: Binary classification for spam detection based on a small vocabulary.

### **1. Computational Constraints: BERT is computationally intensive**

**What does "computationally intensive" mean?**

* **BERT** (Bidirectional Encoder Representations from Transformers) is a powerful and complex model. To work well, it requires a lot of computational power, which means it needs a lot of memory and processing speed.
* **This can be a problem** if you're trying to use BERT in environments with limited resources (like smaller devices or applications that need to work quickly).

**Example**:

* **Real-time inference on edge devices**: Imagine you're trying to use BERT on a **smartphone** or **IoT device**. These devices have limited memory and processing power. BERT, because of its complexity, might be too slow or impossible to run effectively on such devices.
* **Fast predictions**: If you need to make decisions **quickly**—for example, in a **chatbot** that must respond to users within seconds—BERT might take too long to process and generate an answer.

**In short**: BERT needs a lot of computational resources (memory and processing power), so it may not be ideal for tasks or devices that require quick results or have limited hardware.

### **2. Data-Specific Contexts Not Captured in Pre-Trained Models**

**What does this mean?**

* **BERT** is pre-trained on general data (like books, Wikipedia, etc.), so it learns general language patterns. However, it may **miss important details** specific to certain fields or industries (like law or medicine).
* This means BERT might not fully understand or capture the specialized language or nuances used in these fields unless it is **fine-tuned** (trained further on more specific data).

**Example**:

* **Legal or medical datasets**: If you use BERT for tasks related to **law** or **medicine**, it might not understand certain legal or medical terms as well as someone who is specifically trained in those fields.
  + For instance, in a legal document, BERT might confuse the word **"court"** (as in a place where trials happen) with **"court"** (as in a tennis court) because it was trained on general data.
  + In medicine, BERT might not understand complex medical terminology unless it has been specifically trained on a medical corpus.

**In short**: BERT, when used without fine-tuning, may not fully understand specialized language

