
📘 MODULE 2 – WORD AND TEXT REPRESENTATIONS

🧠 1. Word Relationships

Natural Language relies heavily on relationships between words.
Understanding these helps NLP systems find **semantic similarity** (words that mean alike) and **contextual meaning** (words that often appear together).

🔹 A. What Are Word Relationships?

Words are not isolated — their meaning depends on their relation to other words.
Examples:

- "doctor" relates to "hospital", "patient".
- "car" and "automobile" are similar in meaning.
- "hot" is opposite of "cold".

🔹 B. Types of Word Relationships

Type	Description	Example
Synonymy	Words with similar meanings.	*begin–start, big–large*
Antonymy	Opposite meanings.	*good–bad, fast–slow*
Polysemy	Same word, multiple meanings.	*"bank"* → river bank / financial bank
Homonymy	Same spelling/sound, unrelated meanings.	*"bat"* → animal / cricket bat
Hyponymy	"Is-a" relationship (subcategory).	*rose* is a *flower*
Hypernymy	General category relationship.	*fruit → apple, mango*
Collocation	Words that often occur together.	*"make decision", "strong tea"*

🔹 C. The Distributional Hypothesis

“Words that occur in the same contexts tend to have similar meanings.”

Example:

- “dog” and “cat” appear in similar contexts like *pet*, *feed*, *animal*.
- Thus, they should be represented closely in vector space.

This forms the **basis for word embeddings** (Word2Vec, GloVe, etc.).

💡 2. Word Embeddings

🔸 A. What is a Word Embedding?

A **word embedding** represents words as **dense vectors of real numbers** (e.g., 50–300 dimensions).
These vectors are arranged so that **semantically similar words are close** in the vector space.

Word	Simplified 3D Vector
King	[0.8, 0.6, 0.1]
Queen	[0.7, 0.7, 0.2]
Man	[0.9, 0.5, 0.0]
Woman	[0.8, 0.6, 0.1]

👉 Relationship:
[
King - Man + Woman ≈ Queen
]

This is known as **vector arithmetic**, showing that embeddings capture **meaning and analogy**.

🔸 B. Why Word Embeddings?

Earlier methods like **Bag of Words (BoW)** or **TF–IDF** treat words as independent tokens:

- No context.
- No semantic relationship.
- Very high dimensional.

Word embeddings solve these issues by:

- Representing meaning geometrically.
- Being compact and efficient.
- Capturing relationships automatically.

🔸 C. Comparison: Bag of Words vs Word Embedding

Aspect	Bag of Words	Word Embedding
Representation	Count-based	Dense numerical vectors
Context	Ignored	Preserved
Dimension	High (1 per word)	Low (50–300)
Semantics	None	Captured
Example	“dog” ≠ “cat”	“dog” ≈ “cat”

🧩 3. Word Embedding Techniques

⚙️ A. Bag of Words (BoW)

Concept:
Represents a document by the **frequency of each word**, ignoring grammar and order.

Steps:

1. Build a vocabulary of unique words.
2. Count occurrences of each word in every document.

Example:

Sentence	Words
S1: “I love NLP”	I, love, NLP
S2: “I love AI”	I, love, AI

Vocabulary: [I, love, NLP, AI]

Document	I	love	NLP	AI
S1	1	1	1	0
S2	1	1	0	1

Pros:

✅ Simple to understand

✅ Works well for small datasets

Cons:

❌ Ignores order and context

❌ Sparse (large, mostly zero vectors)

⚙️ B. TF–IDF (Term Frequency – Inverse Document Frequency)

Improves BoW by weighting words according to their **importance**.

🔹 Formula

$$
TF(t, d) = \frac{\text{count of term t in document d}}{\text{total words in document d}}
$$

$$
IDF(t) = \log\left(\frac{N}{\text{number of documents containing t}}\right)
$$

$$
TF!-!IDF = TF \times IDF
$$

🔹 Example

Word	Document Frequency	IDF ($\log_{10}$)
the	100	0
machine	5	1.3
learning	3	1.7

If “learning” appears 4 times in one document:

$$
TF–IDF(learning) = 4 \times 1.7 = 6.8
$$

Meaning:
Rare, topic-specific words get higher weight.

Pros:
✅ Highlights meaningful terms
✅ Reduces weight of common words
Cons:
❌ Ignores word order and semantics
❌ High-dimensional sparse matrix

⚙️ **C. Word2Vec**

Developed by **Mikolov et al., 2013 (Google)**
Uses **neural networks** to learn word vectors that encode meaning.

🔹 **Two Architectures**

1. **CBOW (Continuous Bag of Words)**
    - Predicts target word from surrounding context.
    - Example: context = “I __ NLP” → predict “love”.
2. **Skip-Gram**
    - Predicts surrounding context from target word.
    - Example: target = “NLP” → predict (“I”, “love”).

🔹 **How It Works**

Each word is represented by a **one-hot vector** (like [0,0,1,0,0,…]).
The model learns **hidden layer weights** that become dense word vectors.

🔹 **Example in Python (Gensim)**

```
from gensim.models import Word2Vec


sentences = [
    ['machine', 'learning', 'is', 'fun'],
```

```
    ['python', 'makes', 'machine', 'learning', 'easy'],
    ['artificial', 'intelligence', 'includes', 'machine', 'learning']
]

model = Word2Vec(sentences, vector_size=50, window=3, min_count=1, sg=1)

print(model.wv['machine'])  # vector representation
print(model.wv.most_similar('learning'))
```

Output (example):

Vector for 'machine': [0.01, -0.02, 0.03, ...]

Most similar to 'learning': [('python', 0.13), ('intelligence', 0.12)]

🔹 **Optimization Techniques in Word2Vec**

1. **Negative Sampling**
    - Updates only a few weights per training sample.
    - Faster training, works well for large vocabularies.
2. **Hierarchical Softmax**
    - Uses a binary tree to compute probabilities efficiently.
3. **Window Size:**
    - Controls how many surrounding words to consider as context (e.g., 2–5 typical).
4. **Training Method:**
    - CBOW is faster; Skip-Gram works better for rare words.

🔹 **Advantages**

✅ Captures semantic and syntactic relations
✅ Compact representation
✅ Supports analogy reasoning (King–Man+Woman=Queen)

🔹 Limitations

❌ Needs large text corpus
❌ Context window is fixed (doesn't capture long-range meaning)

⚙️ D. Other Embedding Models

Model	Description	Example Use
GloVe (Global Vectors)	Combines global word co-occurrence with local context.	Semantic analysis
FastText	Represents words as subword (character n-gram) vectors → handles misspellings or rare words.	"play", "player", "playing" share features

---

🔢 4. Simple N-Gram Models

🔹 A. Definition

An **N-gram** is a sequence of *N consecutive words* used to model text.

N	Example (Sentence: "I love NLP")
1-gram	I, love, NLP
2-gram	I love, love NLP
3-gram	I love NLP

🔹 B. How N-Gram Models Work

They predict the probability of a word given the previous (N–1) words.

[
P(w_n | w_{n-1}, ..., w_{n-N+1})
]

Example (Bigram Model):
[

P(\text{NLP} | \text{love}) = \frac{Count(love, NLP)}{Count(love)}
]

If:

- Count(love NLP) = 5
- Count(love) = 20
  Then:
  [
  P(\text{NLP} | \text{love}) = 5 / 20 = 0.25
  ]

🔹 **C. Applications**

- Text prediction and autocomplete
- Speech recognition
- Statistical machine translation
- Spelling correction

🔹 **D. Limitations**

- High memory usage for large N.
- Data sparsity (some N-grams may never appear).
- Cannot handle long-range dependencies.

📊 **5. Comparison of All Techniques**

Method	Type	Context Captured	Meaning Captured	Dimensionality
BoW	Count-based	❌ No	❌ No	High
TF–IDF	Weighted count	❌ No	⚠️ Partial	High
Word2Vec	Neural	✅ Yes	✅ Yes	Low
N-Gram	Statistical	✅ Local	⚠️ Partial	Variable

🧭 6. Summary

Concept	Core Idea	Example
Word Relationships	Synonymy, antonymy, and semantic similarity between words.	car–automobile
BoW	Count frequency of words.	[I:1, love:1, NLP:1]
TF–IDF	Weighted frequency to highlight unique words.	“learning”: high weight
Word2Vec	Neural model capturing meaning in dense vectors.	King–Man+Woman=Queen
Optimization	Negative sampling, hierarchical softmax.	Efficient training
N-Gram	Predict next word using context.	“I love → NLP”

✅ **In Summary:**
Word and text representation methods convert language into numbers so that algorithms can analyze meaning and context.
From simple counts (**BoW, TF–IDF**) to neural embeddings (**Word2Vec, GloVe**), and statistical models (**N-gram**), these form the foundation for all modern NLP tasks.