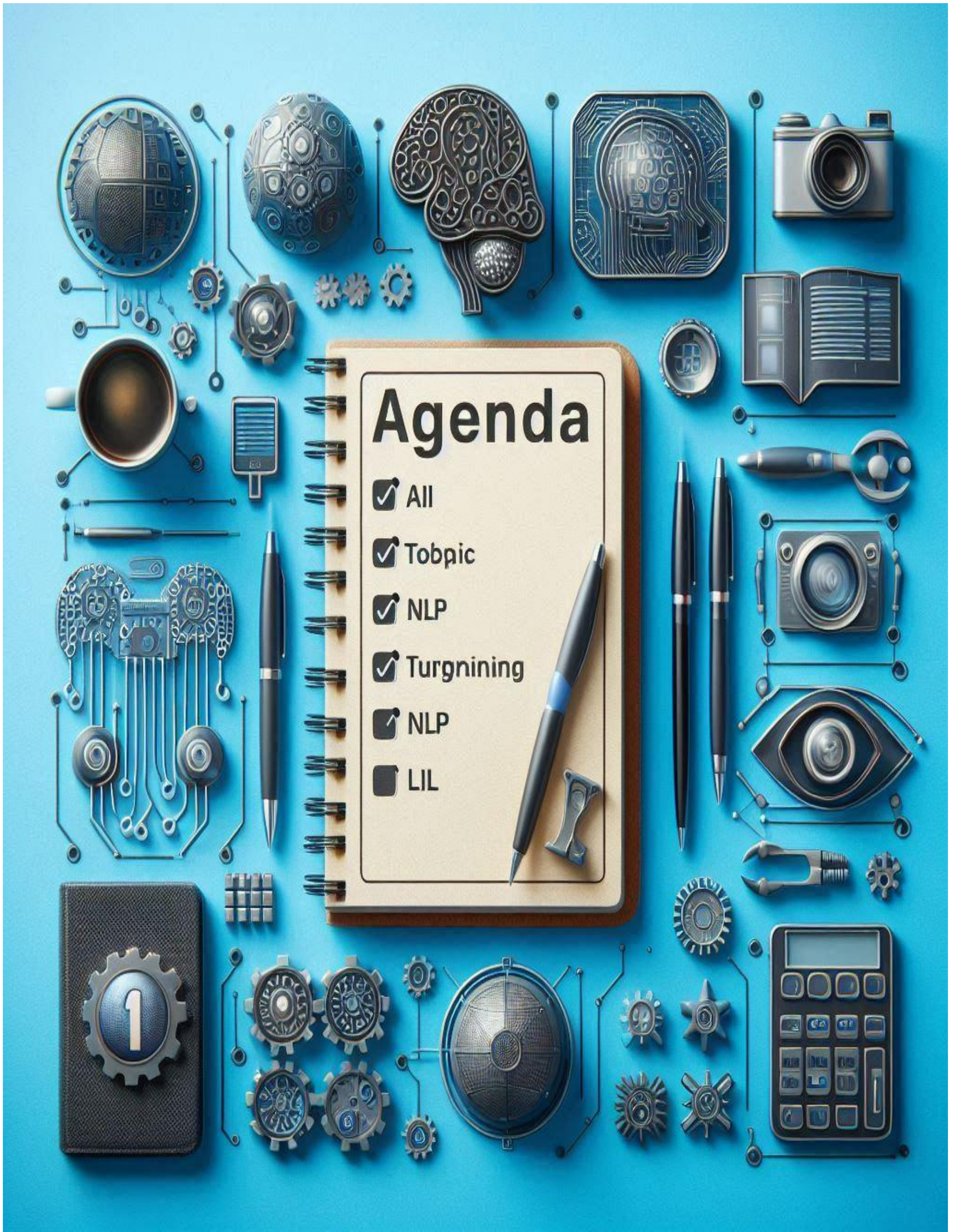
National Telecommunication Institute

What are your expectations for the course?

Agenda



- Lexical Similarity
- Semantic Similarity
- Word & Sentence Embeddings
- Applications
- Evaluation Techniques

What is Text Similarity?

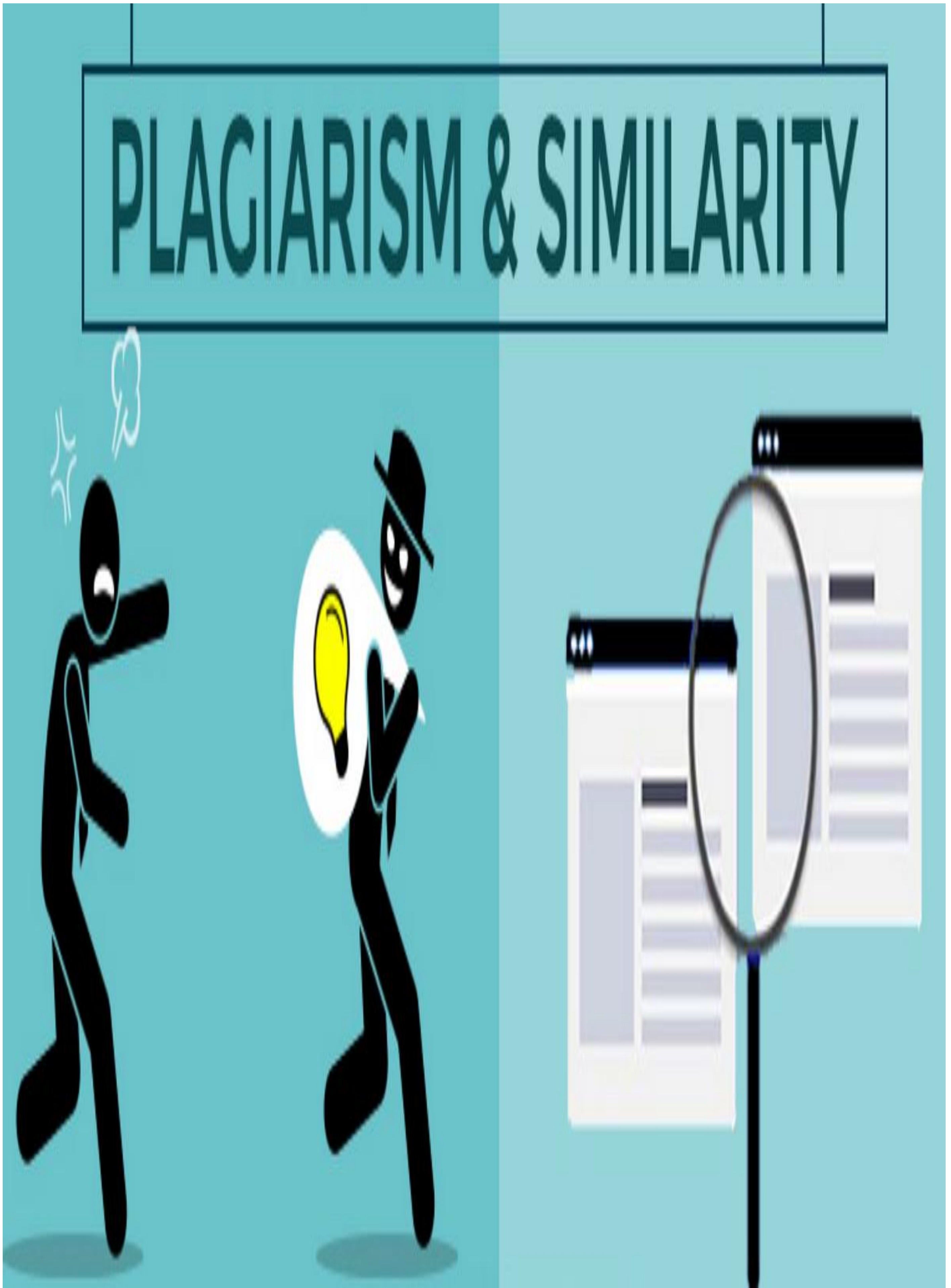
Definition: a Natural Language Processing (NLP) technique that quantifies the degree of similarity between two text elements based on:

- Word content
- Structure
- Meaning

Goals: Quantify how similar two pieces of text are

Purpose: To enable machines to:

- Understand relationships between texts
- Identify duplicates or paraphrases
- Match queries to answers
- Recommend related documents



Real-World Applications

1. Search engines

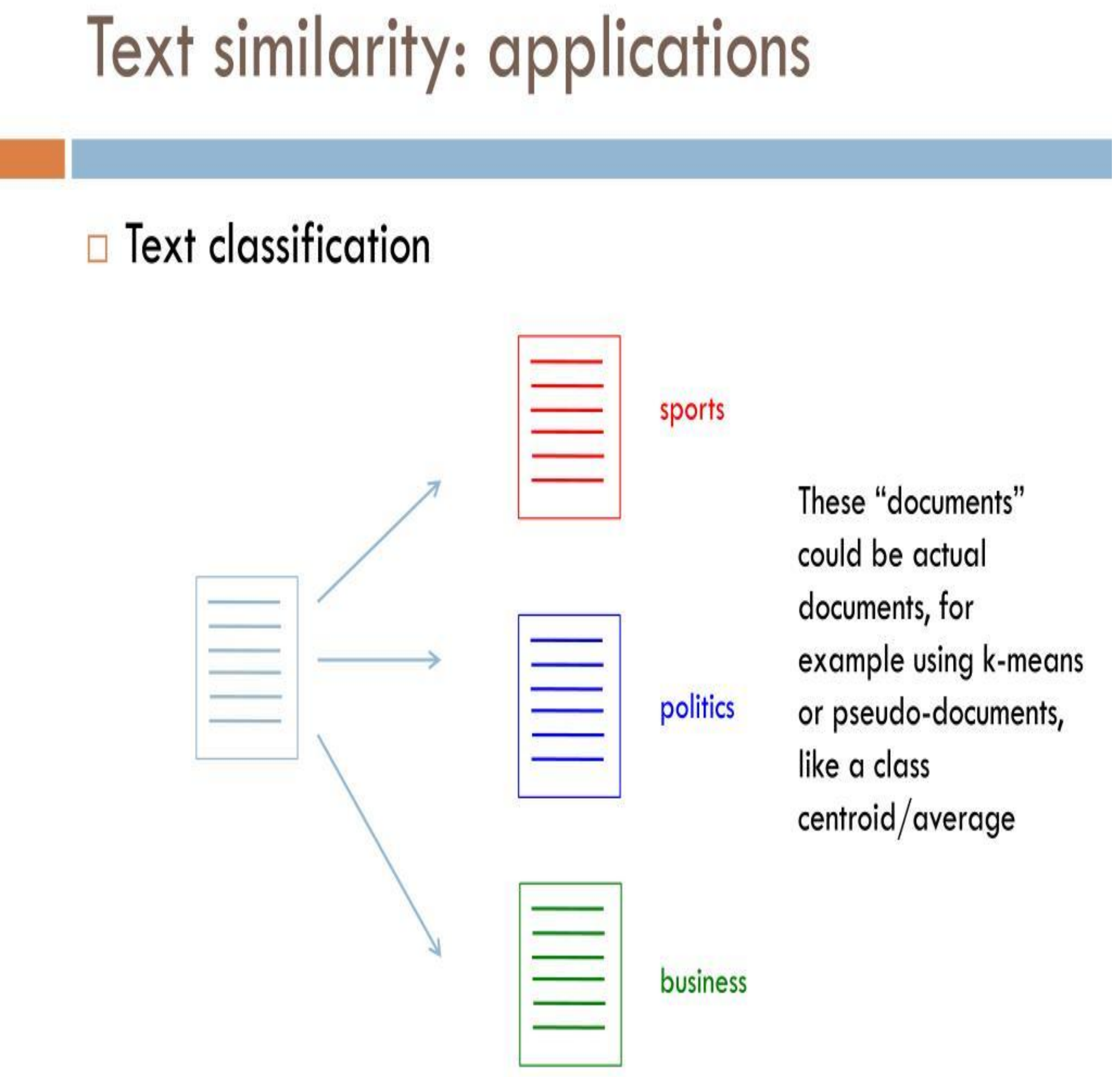
   Help match user queries with the most relevant web pages by comparing textual similarity.
2. Plagiarism detection

   Identify copied or paraphrased content by comparing text similarity between documents.
3. Question answering

   Match a user’s question to the most semantically similar question-answer pair in a database.
4. Duplicate detection

   Detect near-identical entries (e.g., reviews or records) that may differ slightly in wording but convey the same content.

Main Types of Text Similarity



Text similarity: applications

- Text classification

These "documents" could be actual documents, for example using k-means or pseudo-documents, like a class centroid/average

- Lexical Similarity:

  Based on exact word matches (surface-level)

  - example: "car" vs. "cars"

- Semantic Similarity:

  Based on meaning (context-level)

  - example: " I love dogs" vs. "I adore canines"

What is Lexical Similarity?

Lexical similarity is a measure that compares texts based on **surface-level features** like:

- Exact word matches
- Spelling overlap
- Common sub sequences

Works purely on form, not meaning

Sensitive to spelling and word order

Simple and fast to compute

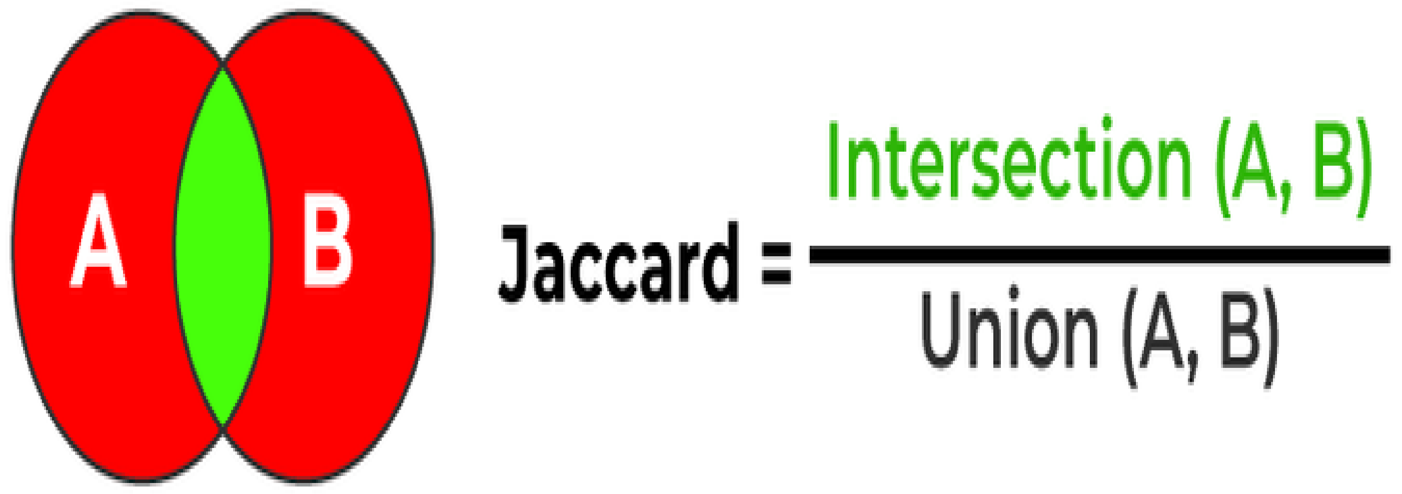
Common Techniques

- **Jaccard Similarity** – compares common vs total words
- **Cosine Similarity** – compares text as vectors
- **Edit Distance (Levenshtein)** – counts character changes

Limitations

- Doesn’t capture synonyms or paraphrasing
- “*I love cats*” and “*I adore felines*” are lexically different but semantically similar

Jaccard Similarity



$$\text{Jaccard} = \frac{\text{Intersection }(A, B)}{\text{Union }(A, B)}$$

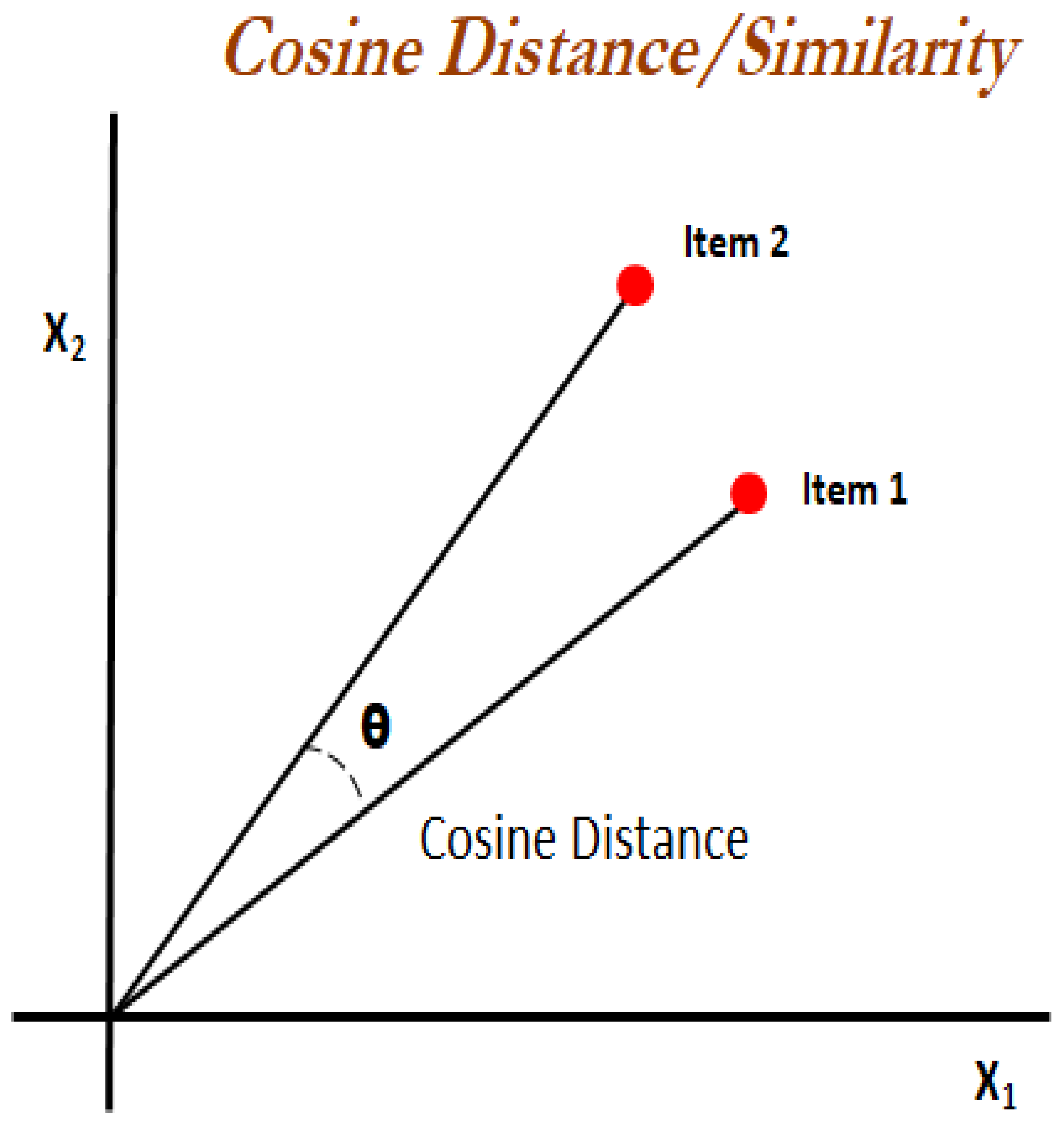
- a simple statistical method used to measure the **similarity between two sets**, often applied to text by comparing sets of words or tokens.
- It measures **how many words two texts share**, divided by **how many total unique words they have**.
- **Advantages:**

Simple and intuitive
Useful for short, bag-of-words comparisons

- **Limitations:**

Doesn’t account for word meaning
Sensitive to different wording or synonyms

What is Cosine Similarity?



- A metric that measures the **cosine of the angle** between two text vectors in a multidimensional space — it tells us **how similar two texts are in direction**, regardless of their length.
- It compares **word frequency patterns** rather than exact words — the more similar the patterns, the **smaller the angle**, and the **closer to 1** the similarity.
- **Advantages:**
  - Ignores text length
  - Works well with high-dimensional sparse data
  - Common in information retrieval (search engines)
- **Limitations:**
  - Still lexical — doesn't fully understand word meanings
  - Relies on quality of vectorization (e.g., BoW or TF-IDF)

Most Common Clustering Models in NLP

Edit Distance, also known as **Levenshtein Distance**, is a metric that calculates the **minimum number of operations** needed to transform one string into another.

1. **Insertion** – adding a character
2. **Deletion** – removing a character
3. **Substitution** – changing one character to another

•**How different two strings are** based on spelling and character-level changes

•Lower distance → more similar

Advantages:

Very useful for **spelling correction**, **fuzzy matching**, and **typo detection**

Limitations:

Works only at **character level**

Doesn't capture **semantic similarity** or word meaning

Limitations of Lexical Similarity Methods

1. **No Understanding of Meaning**
   •Treats words as isolated symbols
   •Cannot detect synonyms or paraphrased expressions
2. **Sensitive to Word Forms**
   - Small variations (like plurals or verb tenses) reduce similarity
3. **Ignores Word Order & Context**
   - Bag-of-Words and related models ignore grammar and structure
4. **Doesn’t Handle Polysemy**
   - Words with multiple meanings aren’t disambiguated

What is Semantic Similarity?

- a measure of how much **two pieces of text share in meaning**, regardless of the actual words used.
- It evaluates whether two texts express **the same idea or concept**, even if they use **different words or phrases**.
- Goes beyond surface-level word matching
- Uses **context**, **word meaning**, and **embedding models**
- Recognizes **synonyms**, **paraphrases**, and **related ideas**

How it work?

Work based on:

- **Word meanings (semantics)**
- **Word relationships** (e.g., synonyms, hypernyms)
- **Contextual usage** in sentences

Advanced:

- More accurate for understanding human language
- Works well for applications like:
  - Search engines
  - Chatbots
  - Paraphrase detection
  - Semantic search

Challenges:

- Requires **more computation and data**
- Needs **pre-trained models** or knowledge bases
- May struggle with ambiguous or informal language

Word Meaning and Context

Understanding a word’s meaning **requires considering the context in which it appears** — this is fundamental to **semantic analysis**.

Context refers to the **surrounding words, sentence structure, and topic** that influence a word's meaning.

Why It's Important:

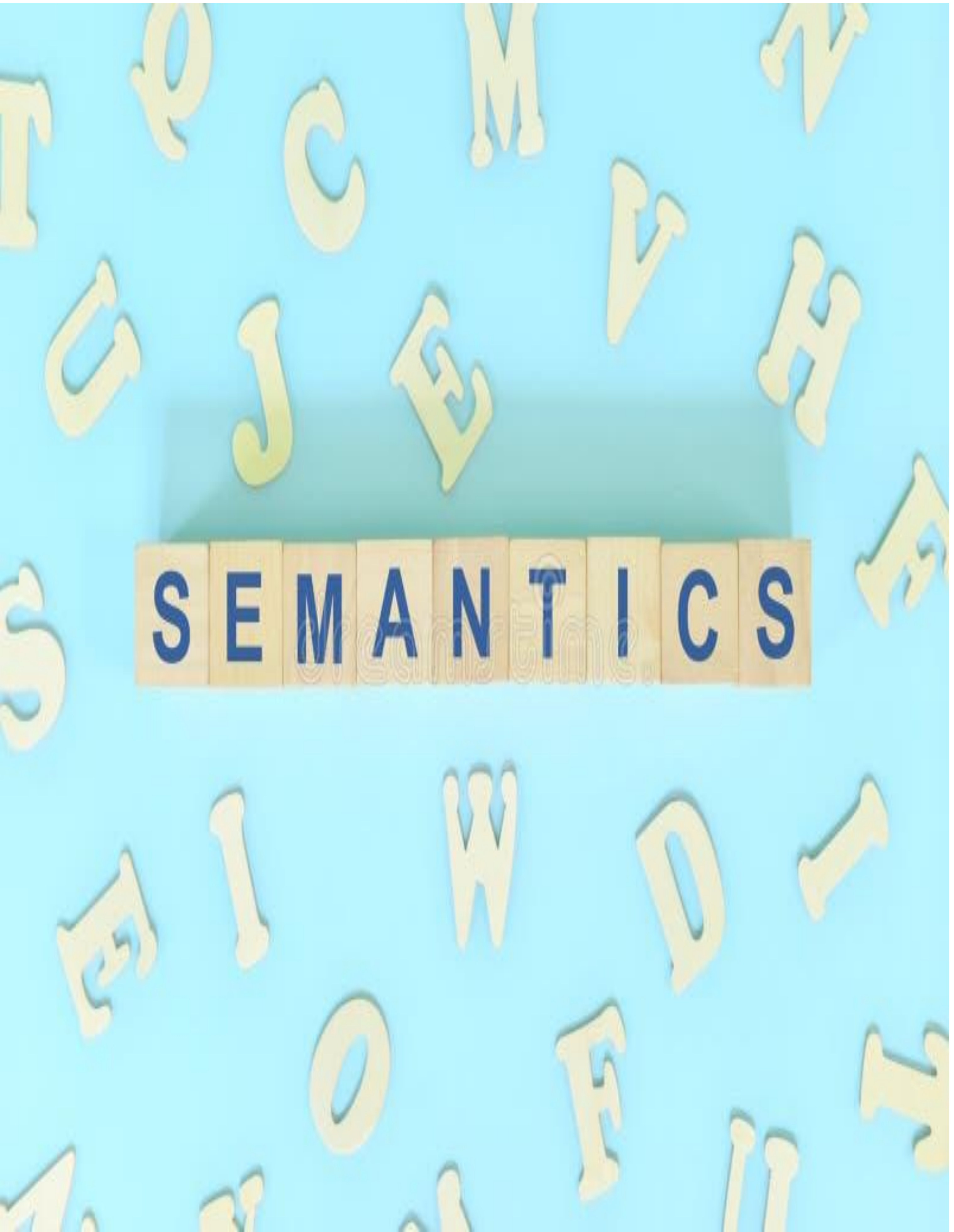
•Many words have **multiple meanings** (polysemy)

Example: “bat” (animal) vs “bat” (sports equipment)

•The correct interpretation **depends on context**

“She swung the bat” → sports

“The bat flew away” → animal



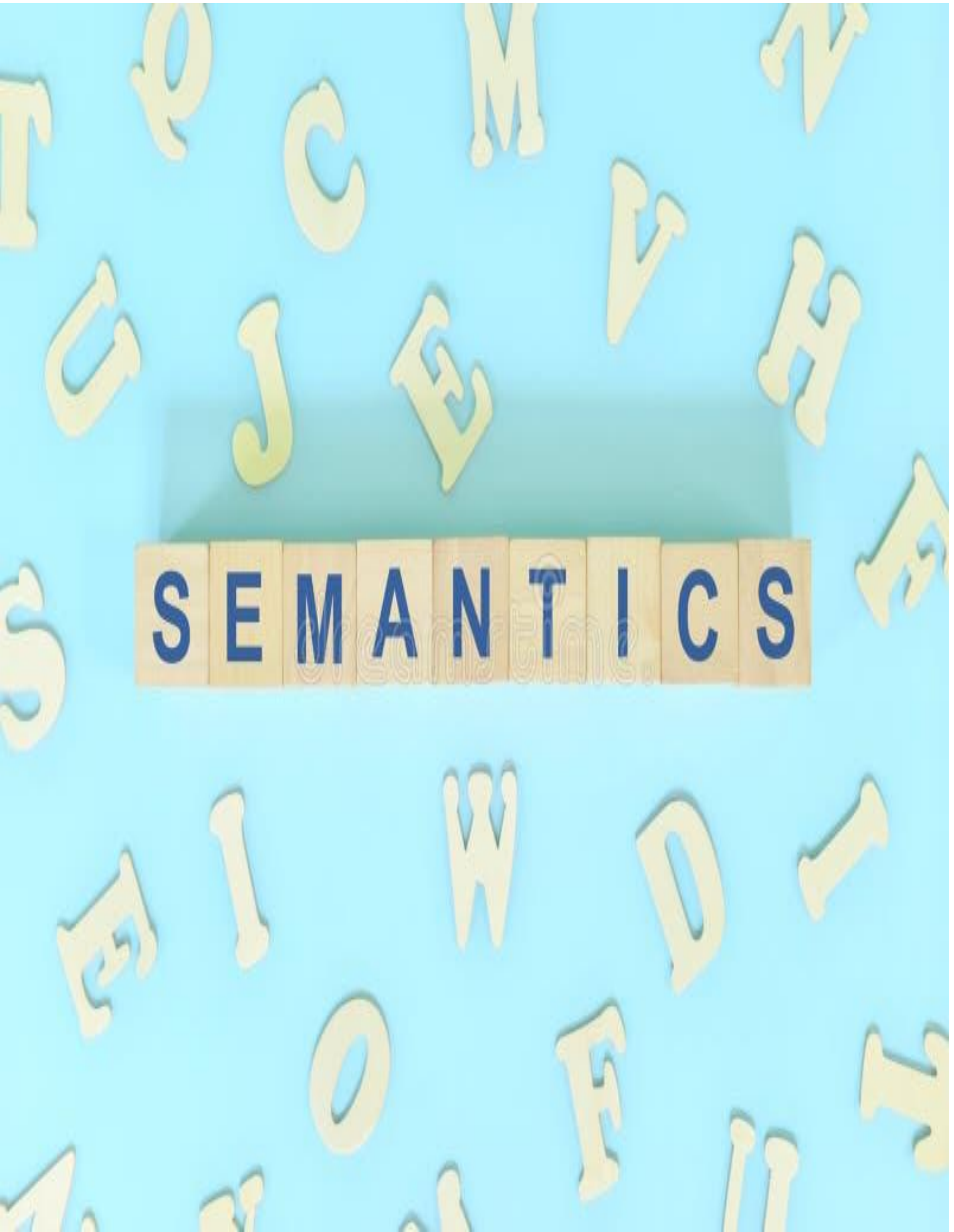
What is Semantics in NLP?

Semantics plays a critical role in **enabling machines to understand the *meaning* behind human language**, rather than just processing raw text.

It refers to interpreting **what words and sentences actually mean** in context — not just recognizing their form.

Why It's Important?

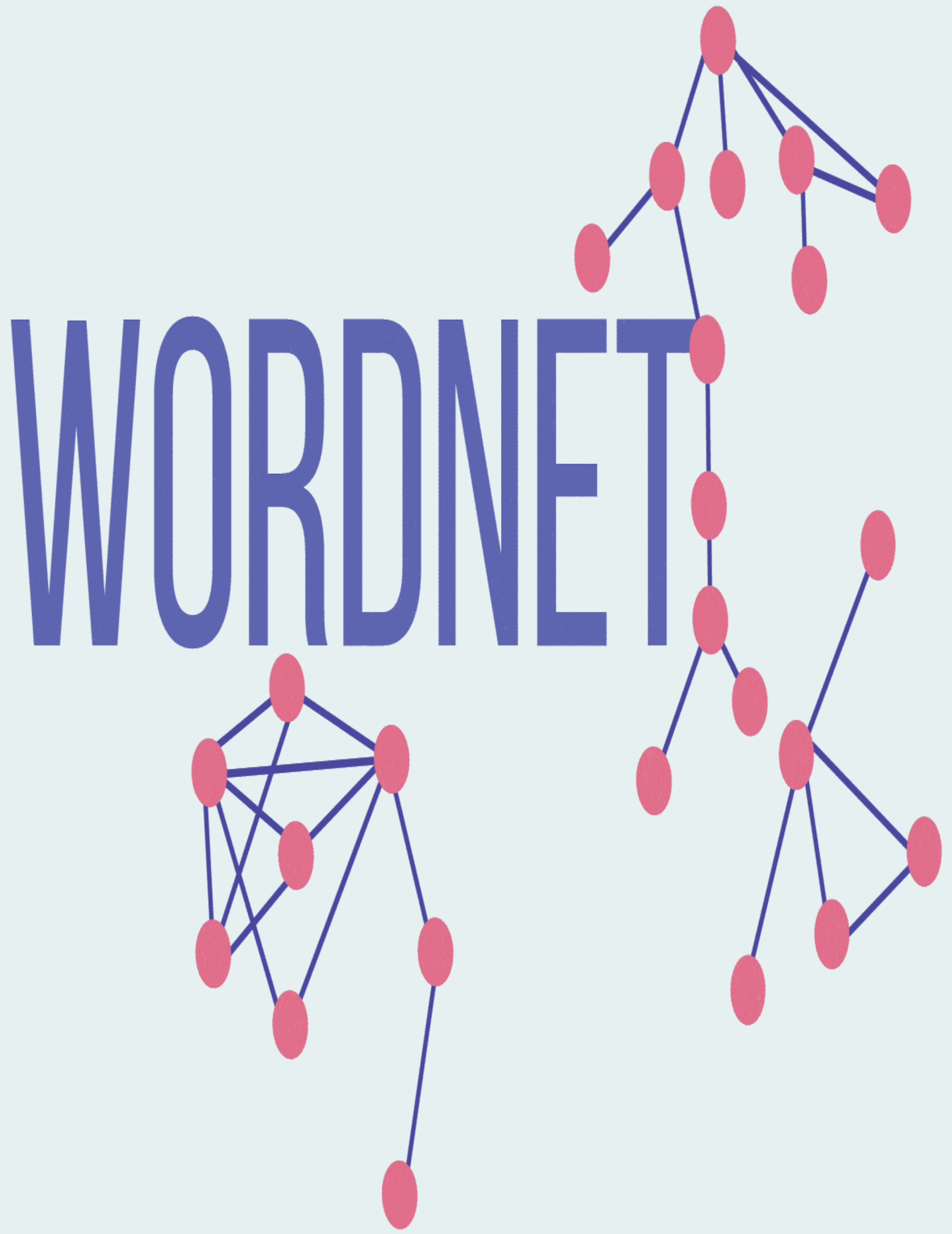
1. Disambiguating Word Meanings
2. Understanding Paraphrases
3. Interpreting Intent and Sentiment
4. Answering Questions Accurately



Tools for Semantic Analysis

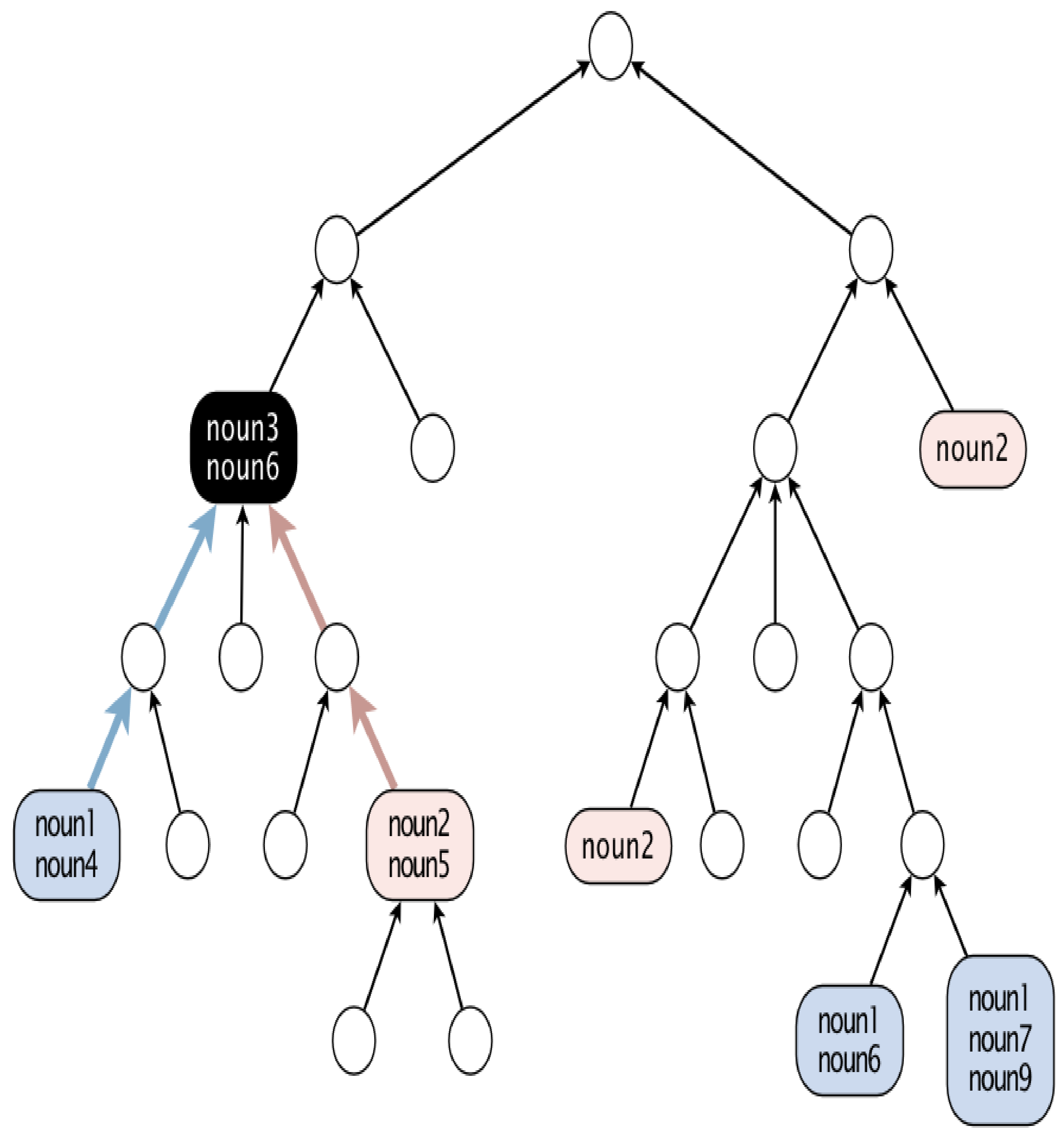
1. **WordNet:**A large lexical database of English that organizes words into synsets (sets of synonyms)
2. **Word Embeddings:** Words are represented as dense vectors that capture meaning based on context.
3. **Sentence Embeddings** sa secnetnes ertine tneserpeR : .gninaem llarevo tcefler ot srotcev
4. **Transformer-based Language Models** erawa-txetnoC : .txetnoc ni gninaem dnatsrednu taht sledom
5. **SpaCy & NLTK :**Useful for combining structure and meaning

Introduction to WordNet



- **WordNet** is a large lexical database of English that organizes words into sets of **cognitively meaningful groups** and links them through **semantic relationships**.
- WordNet combines elements of **a dictionary and a thesaurus**. It groups words that express the same concept into **synsets** (synonym sets), each representing one meaning or sense.

Structure of WordNet



distance(noun1, noun2) = 4

sca(noun1, noun2) = {noun3, noun6}

Synsets: Groups of synonyms sharing the same meaning

Example: *car, automobile, auto* = one synset

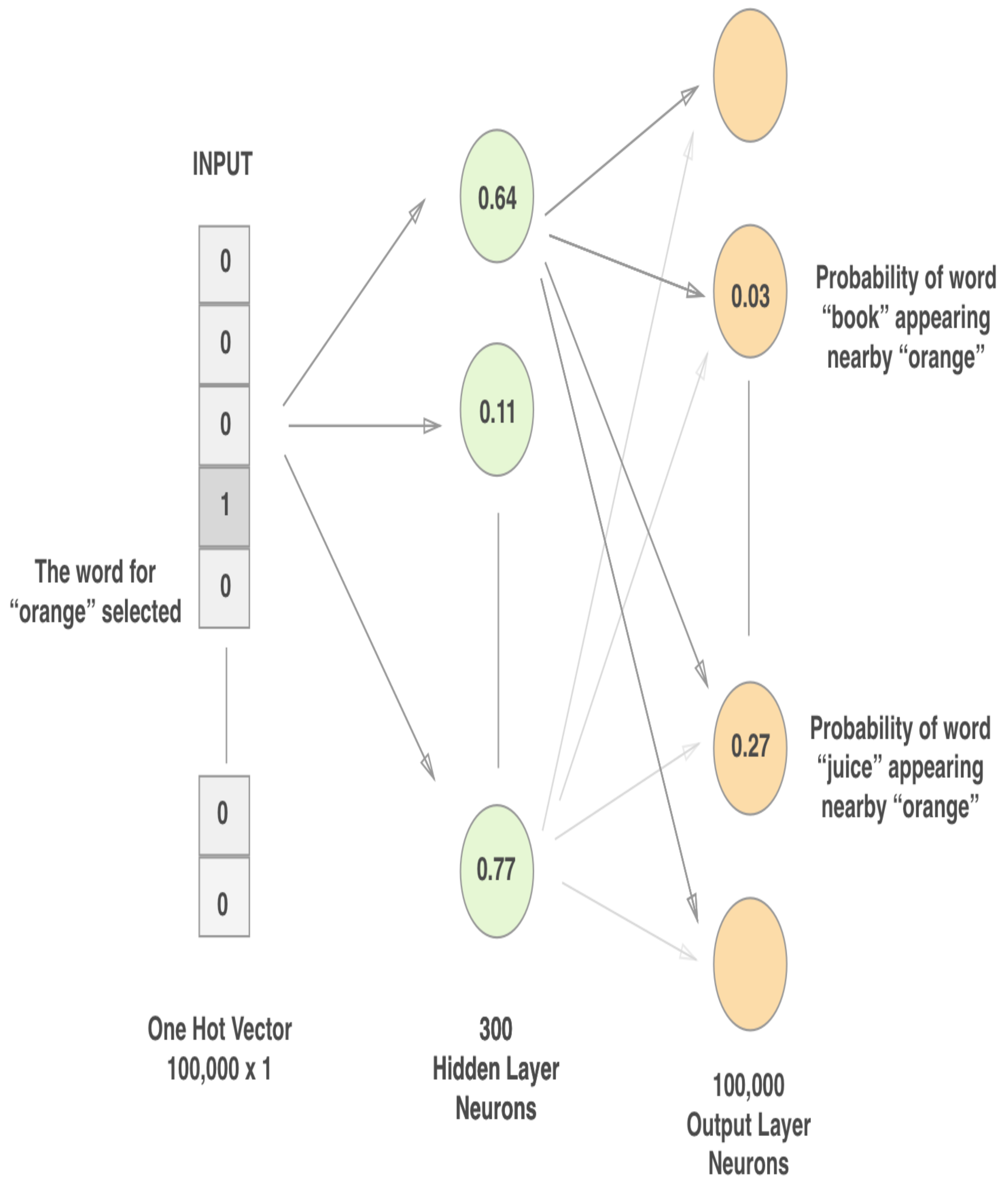
Each synset includes:

- A **definition** (gloss)
- One or more **example sentences**

Why wordnet:

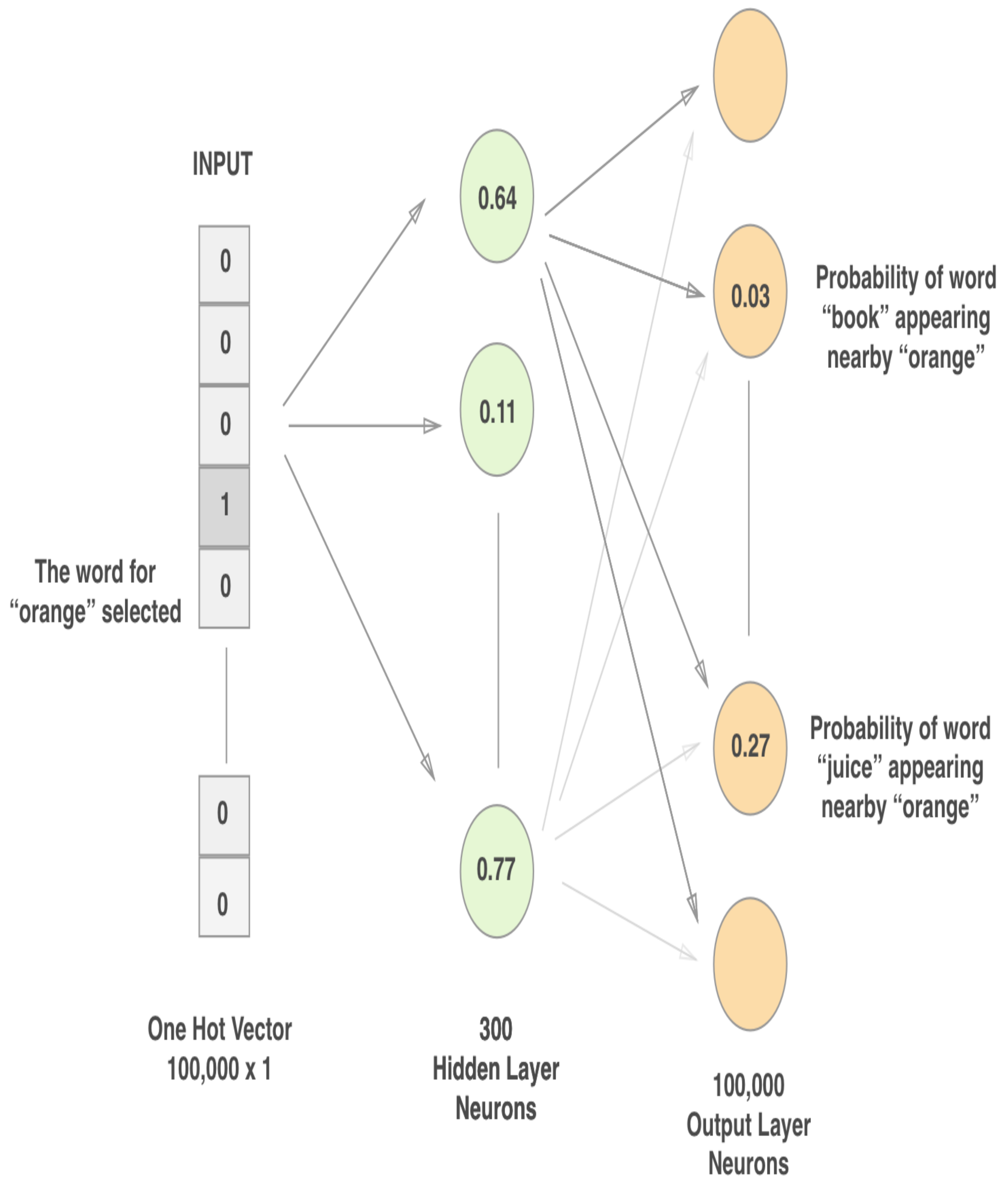
- Clarifies **word meaning and sense**
- Supports **word similarity** and **disambiguation**
- Useful for **semantic search**, **paraphrasing**, and **text summarization**

Introduction to Word Embeddings


INPUT
0
0
0
1
0
0
0
The word for "orange" selected
One Hot Vector 100,000 x 1
0.64
0.11
0.77
300 Hidden Layer Neurons
0.03
0.27
Probability of word "book" appearing nearby "orange"
Probability of word "juice" appearing nearby "orange"
100,000 Output Layer Neurons

- **Word embeddings** are a way of representing words as **dense numerical vectors** in a continuous vector space, where **semantically similar words are placed closer together**.
- Instead of representing words as one-hot vectors (which are sparse and don't capture meaning), embeddings encode words as real-valued vectors that **reflect their meaning based on usage**.
- They are learned from large text corpora using neural models.
- Words that appear in **similar contexts** tend to have **similar meanings**, and therefore **similar vector representations**.

What Word Embeddings Capture


INPUT
0
0
0
1
0
0
0
The word for "orange" selected
One Hot Vector 100,000 x 1
0.64
0.11
0.77
300 Hidden Layer Neurons
0.03
0.27
Probability of word "book" appearing nearby "orange"
Probability of word "juice" appearing nearby "orange"
100,000 Output Layer Neurons

- **Semantic similarity**: *happy ↔ joyful*
- **Syntactic patterns**: *king - man + woman ≈ queen*
- **Analogies and word relationships**

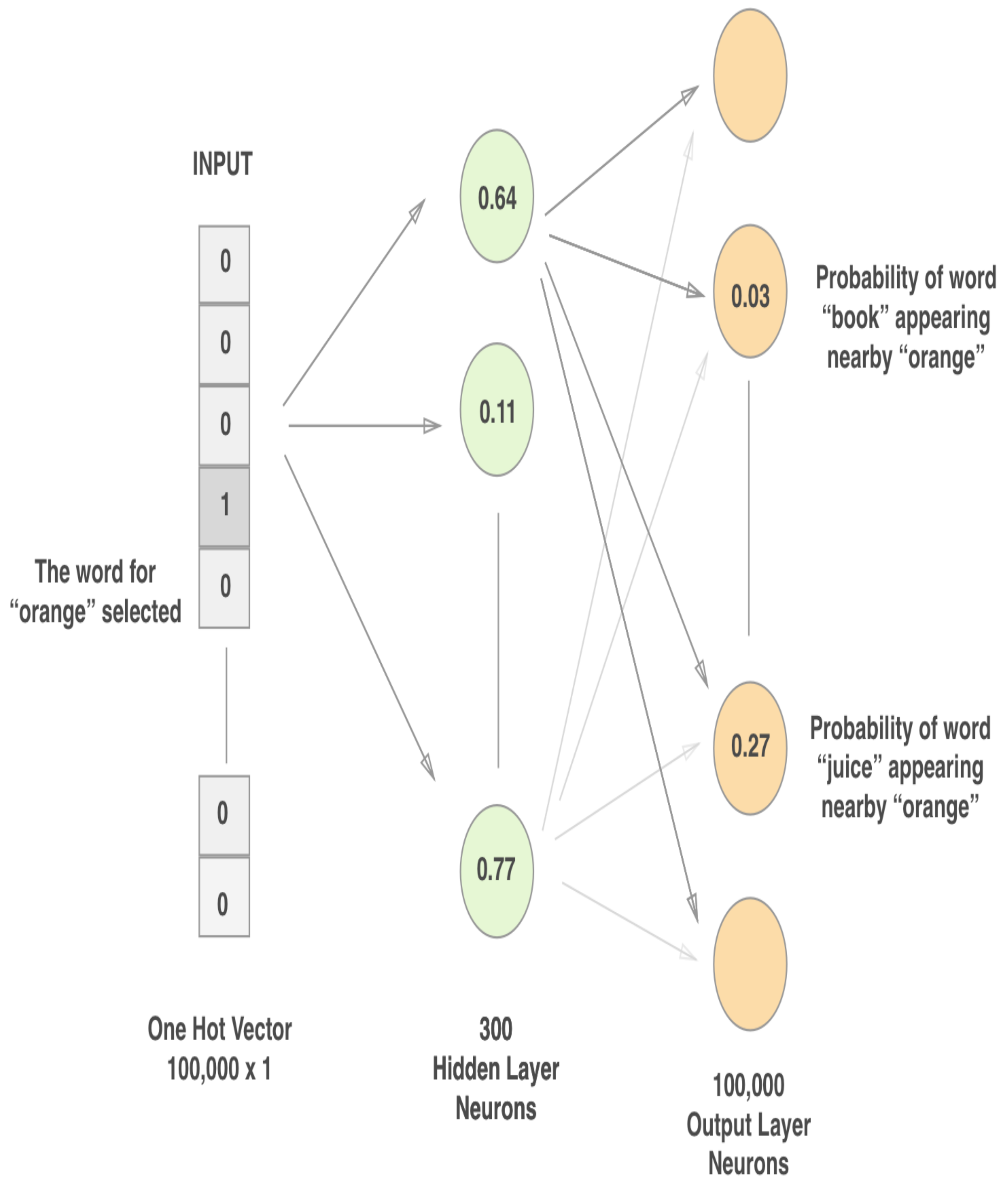
Vector Space Example:

Words like "Paris", "London", "Rome" are clustered together

Words like "dog", "cat", "rabbit" form another cluster

National Telecommunication Institute

Benefits of Word Embeddings


INPUT
0
0
0
1
0
0
0
The word for "orange" selected
0.64
0.11
0.77
0.03
0.27
Probability of word "book" appearing nearby "orange"
Probability of word "juice" appearing nearby "orange"
One Hot Vector 100,000 x 1
300 Hidden Layer Neurons
100,000 Output Layer Neurons

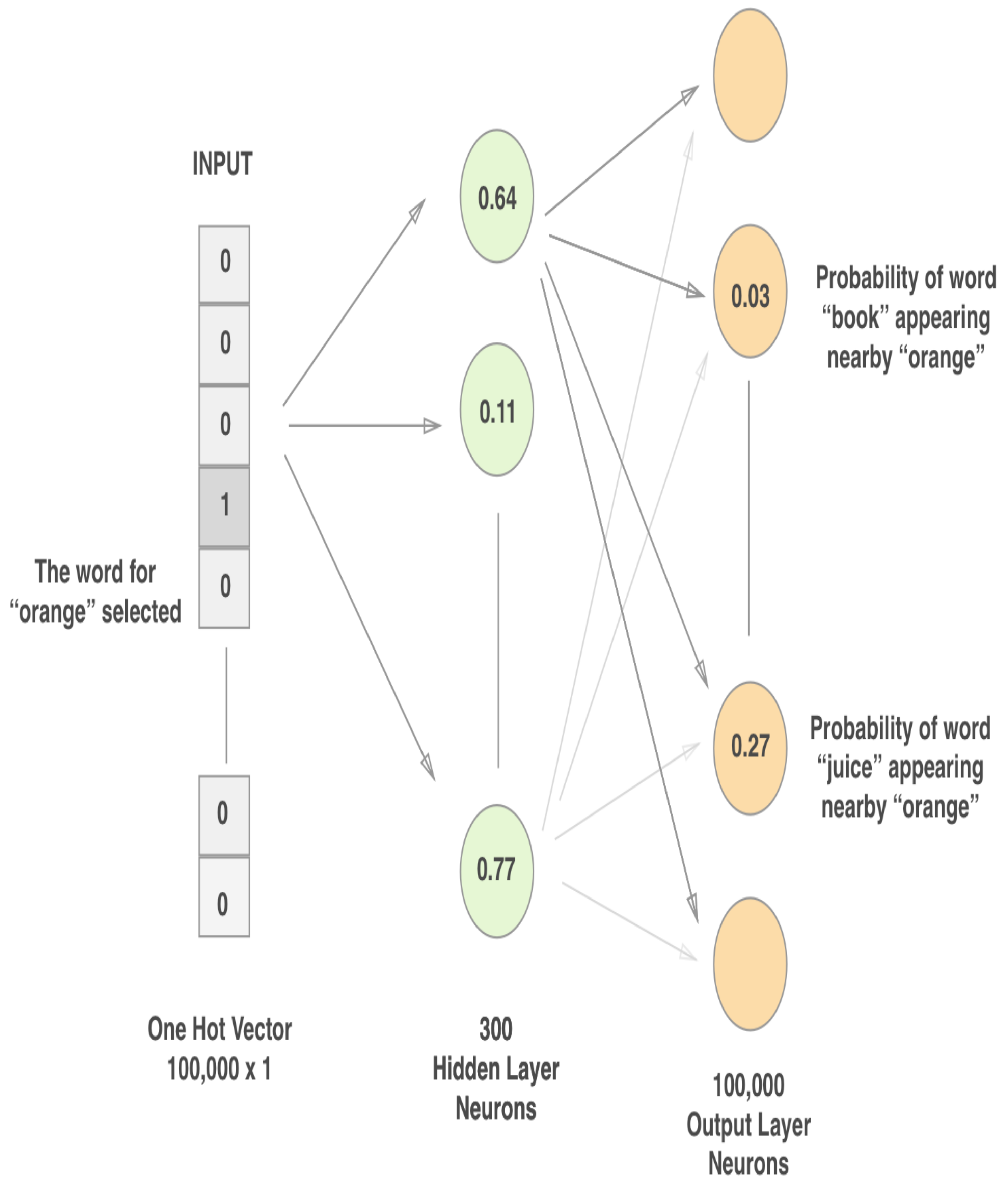
Capture both meaning and context

Allow mathematical comparison of words

Improve performance in NLP tasks like:

- Sentiment analysis
- Machine translation
- Text classification

Types of Word Embeddings


INPUT
0
0
0
1
0
0
0
The word for "orange" selected
0.64
0.11
0.77
0.03
0.27
Probability of word "book" appearing nearby "orange"
Probability of word "juice" appearing nearby "orange"
One Hot Vector 100,000 x 1
300 Hidden Layer Neurons
100,000 Output Layer Neurons

- **Word2Vec** – based on context prediction (CBOW, Skip-gram)

  Example: *king – man + woman ≈ queen*

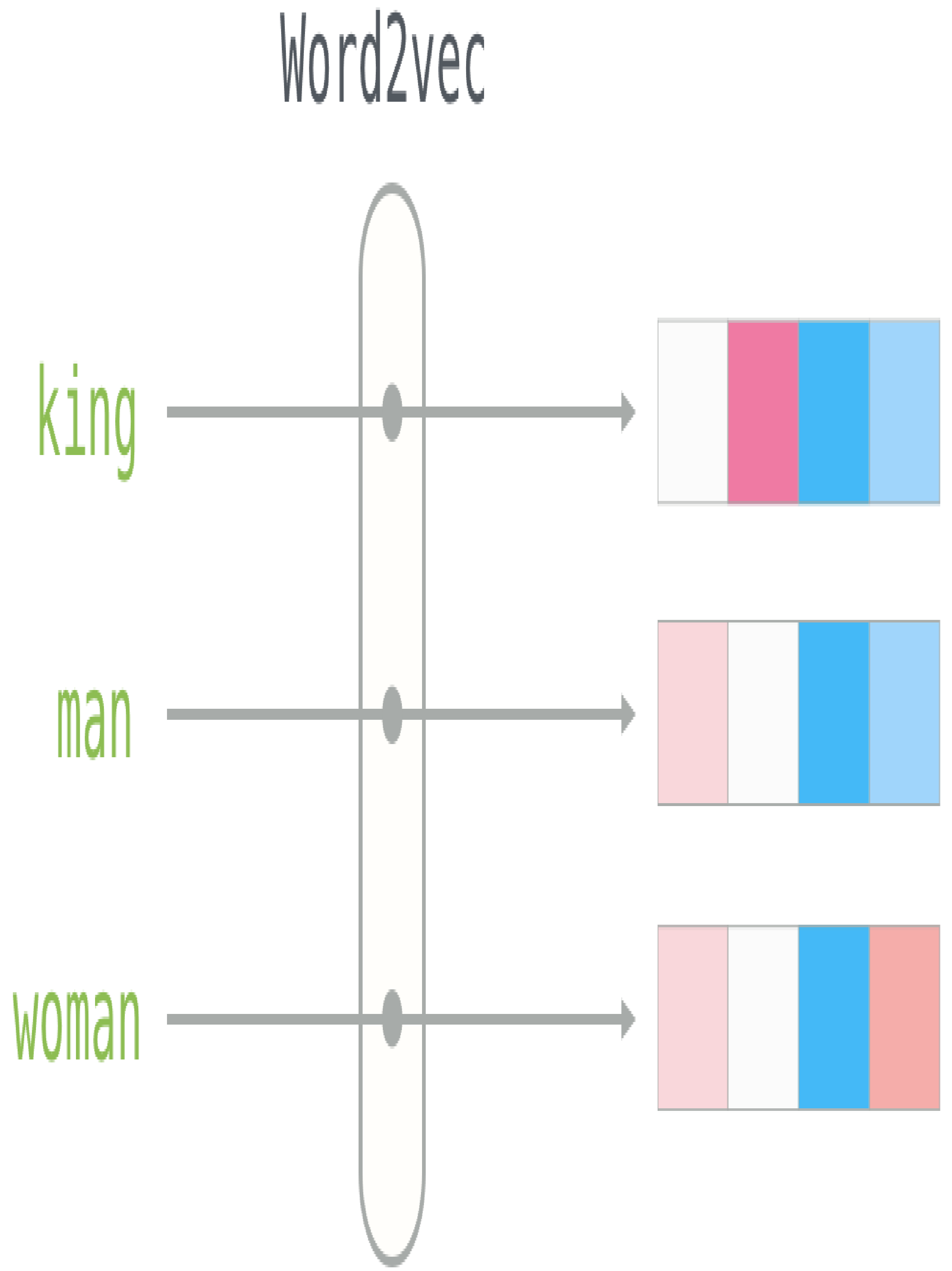
- **GloVe** – based on global co-occurrence

  Example: *ice* and *cold* are close due to frequent co-occurrence

- **FastText** – adds subword information for rare words

  Example: "reading" shares components with "read", "reader", "reads"

Word2Vec



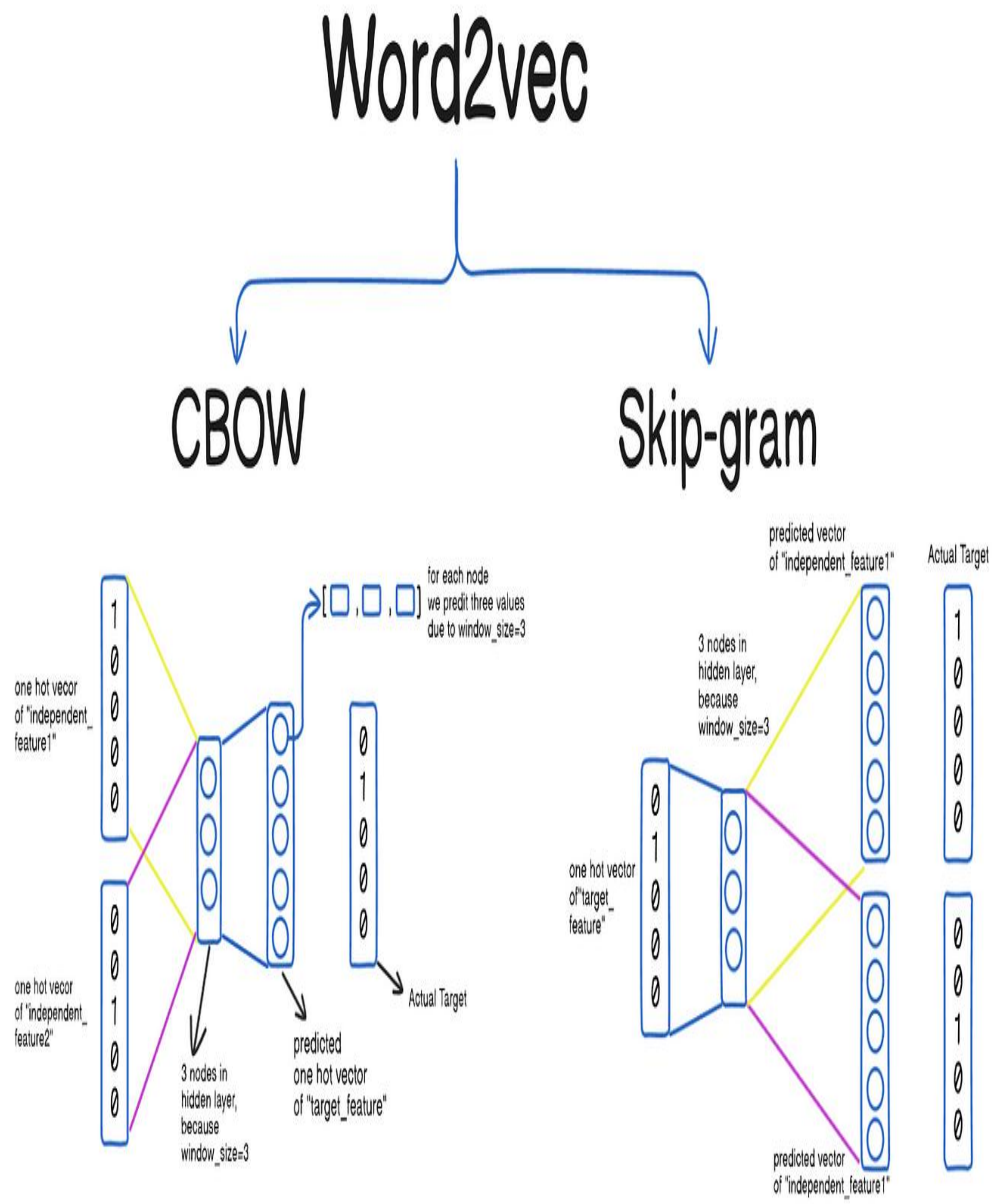
- A powerful word embedding model developed by Google that learns the **meaning of words** by analyzing the **context in which they appear**.
- Word2Vec represents words as **vectors in a continuous space**, trained using **neural networks** based on surrounding words.

Learns from Word Context:

Words that **appear in similar contexts** get similar vector representations

Example: *king*, *queen*, *prince*, *princess* will be close together in the vector space

Architectures and Benefits


Word2vec
CBOW
Skip-gram
one hot vecor of "independent_feature1"
one hot vecor of "independent_feature2"
for each node we predit three values due to window_size=3
3 nodes in hidden layer, because window_size=3
predicted one hot vector of "target_feature"
Actual Target
predicted vector of "independent_feature1"
Actual Target
3 nodes in hidden layer, because window_size=3
one hot vector of "target_feature"
predicted vector of "independent_feature1"

1. CBOW (Continuous Bag of Words)

Predicts a **target word** from its surrounding **context words**

Example: Given "I ___ cats", it tries to predict "love"

2. Skip-gram

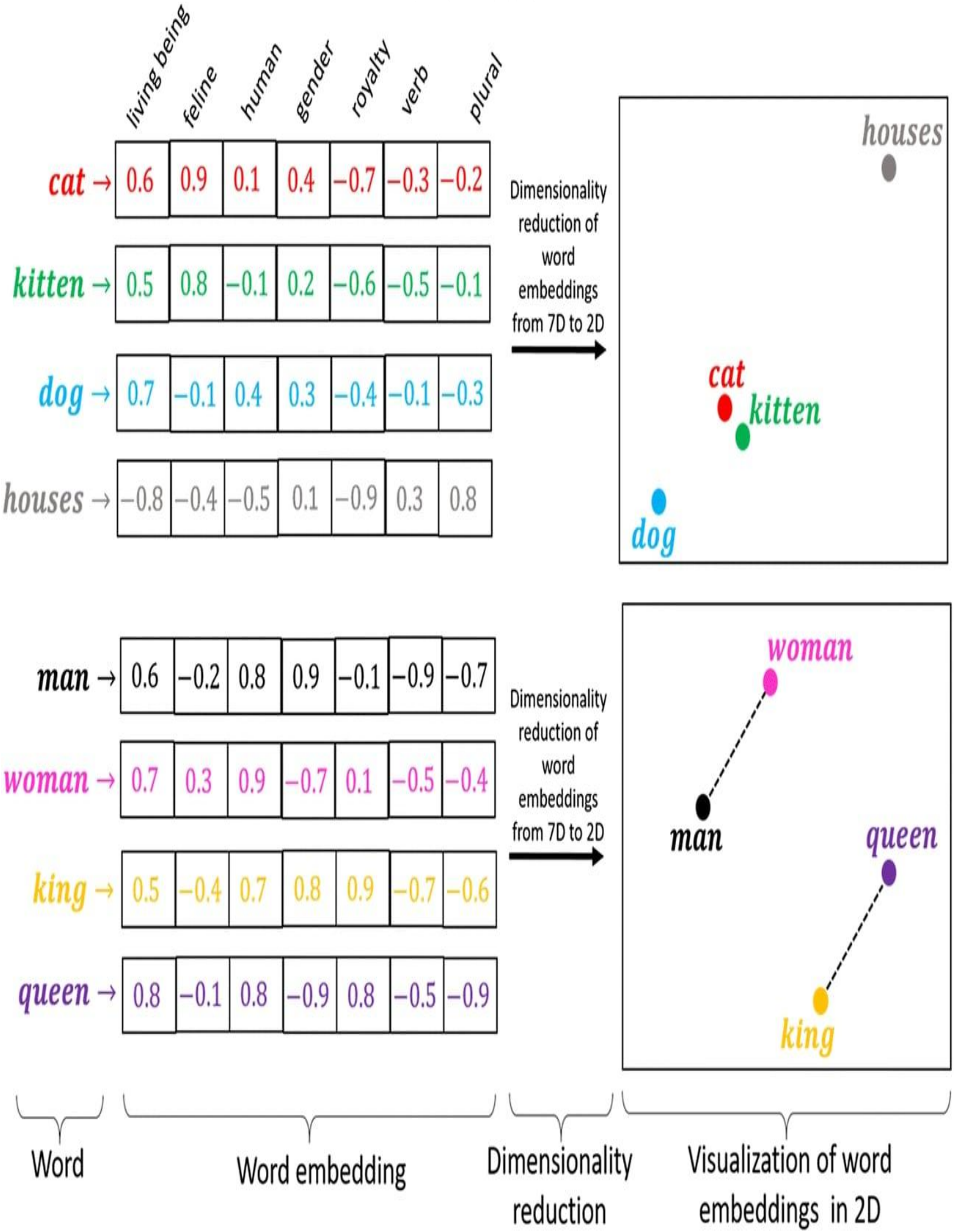
Predicts **context words** from a given **target word**

Example: Given "love", it tries to predict "I" and "cats"

Benefits:

- Captures both **syntactic** (grammar) and **semantic** (meaning) relationships
- Lightweight and fast to train
- Forms the foundation for many modern NLP systems

GloVe (Global Vectors for Word Representation)



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
Dimensionality reduction of word embeddings from 7D to 2D
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

GloVe is a word embedding model developed by **Stanford** that combines the benefits of **global co-occurrence statistics** with the efficiency of vector-based representation.

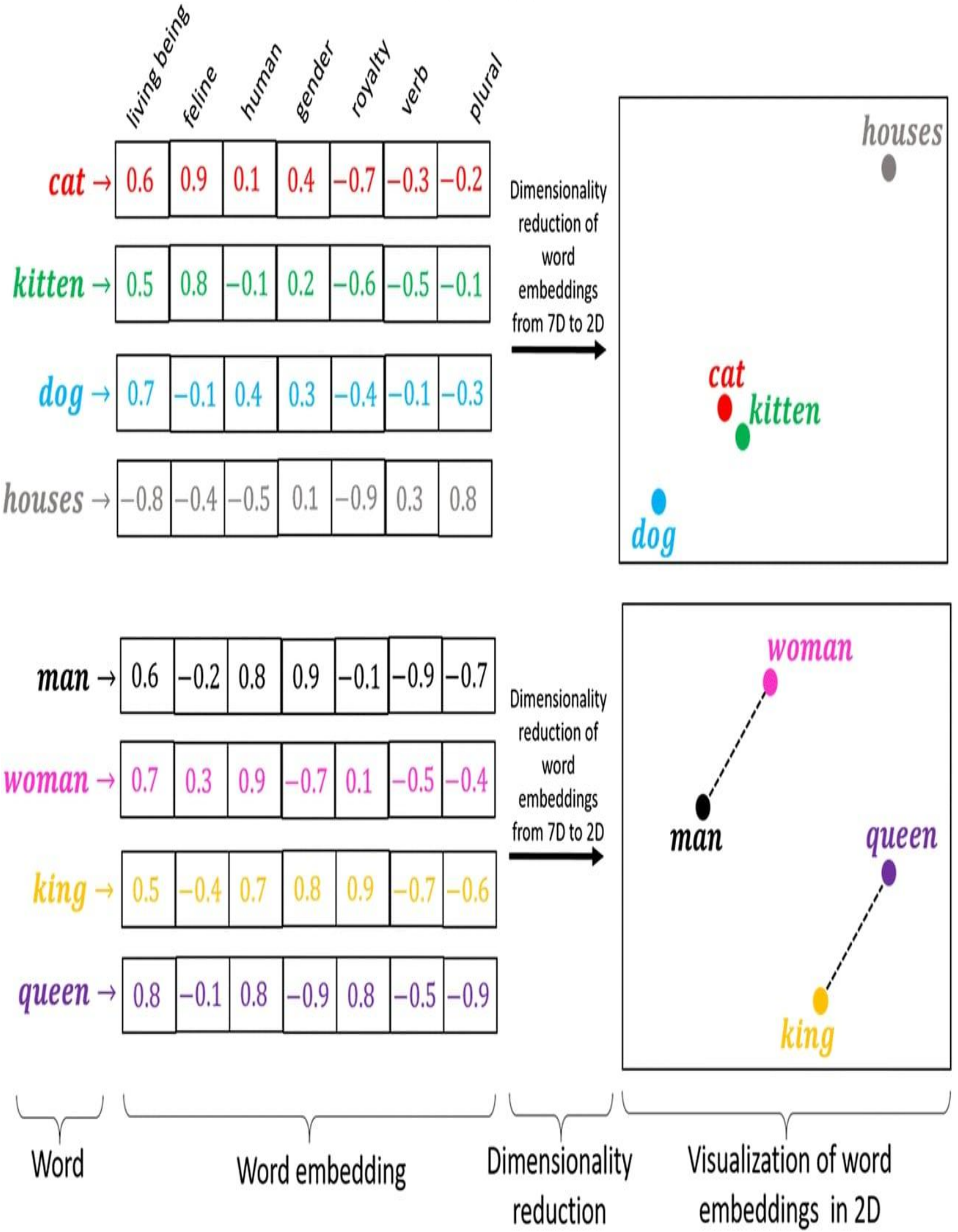
Words that occur in similar **global contexts** tend to have similar meanings.

GloVe captures these patterns using a **word-word co-occurrence matrix** across the entire corpus.

Common Use Cases:

- Semantic similarity
- Named entity recognition
- Text classification

How It Works & Benefits


living being
feline
human
gender
royalty
verb
plural
cat → 0.6 0.9 0.1 0.4 −0.7 −0.3 −0.2
kitten → 0.5 0.8 −0.1 0.2 −0.6 −0.5 −0.1
dog → 0.7 −0.1 0.4 0.3 −0.4 −0.1 −0.3
houses → −0.8 −0.4 −0.5 0.1 −0.9 0.3 0.8
man → 0.6 −0.2 0.8 0.9 −0.1 −0.9 −0.7
woman → 0.7 0.3 0.9 −0.7 0.1 −0.5 −0.4
king → 0.5 −0.4 0.7 0.8 0.9 −0.7 −0.6
queen → 0.8 −0.1 0.8 −0.9 0.8 −0.5 −0.9
Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Builds a matrix where each cell shows how often word *i* appears with word *j*

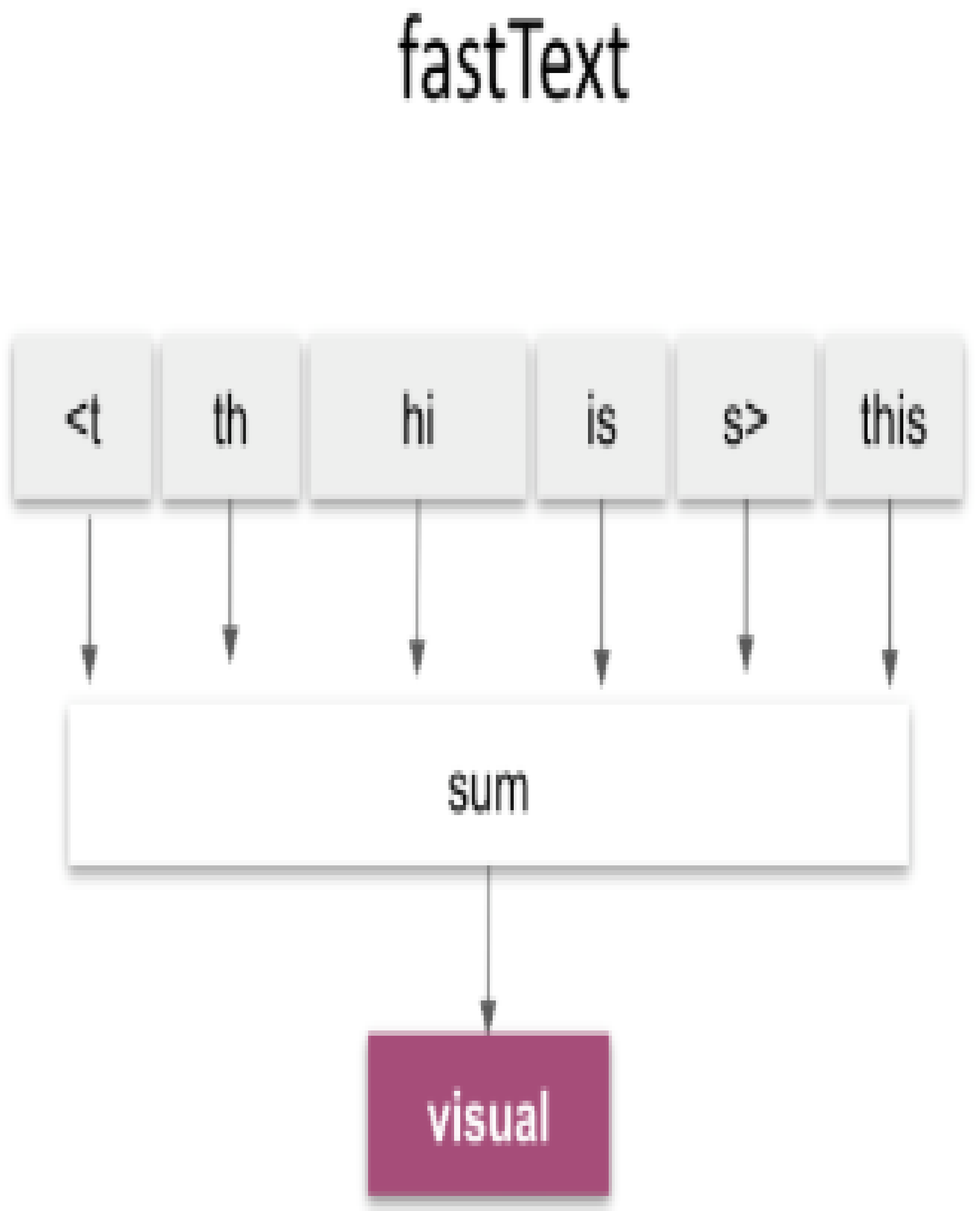
Learns word vectors that preserve **ratios of co-occurrence** probabilities

Embeddings are optimized so that similar words have similar vectors

Benefits of GloVe:

- Captures **global relationships**, unlike Word2Vec which focuses on local context only
- Produces **pre-trained embeddings** from huge corpora like Wikipedia
- Embeddings reflect **meaningful distances and directions** (e.g., gender or tense)

FastText



FastText is a word embedding model developed by **Facebook AI** that improves upon Word2Vec by representing words using **character-level information**.

Instead of treating each word as a unique token, **FastText breaks words into subword units** (character n-grams) and builds embeddings from these smaller pieces.

How It Works:

- A word like "playing" is broken into:
  - <pla, lay, ayi, yin, ing>
- The word's vector is computed as the **sum of its n-gram vectors**

Why We Need Sentence-Level Representations:

•Sentences express **complete ideas**, not just word lists

Word meaning changes depending on **surrounding words**

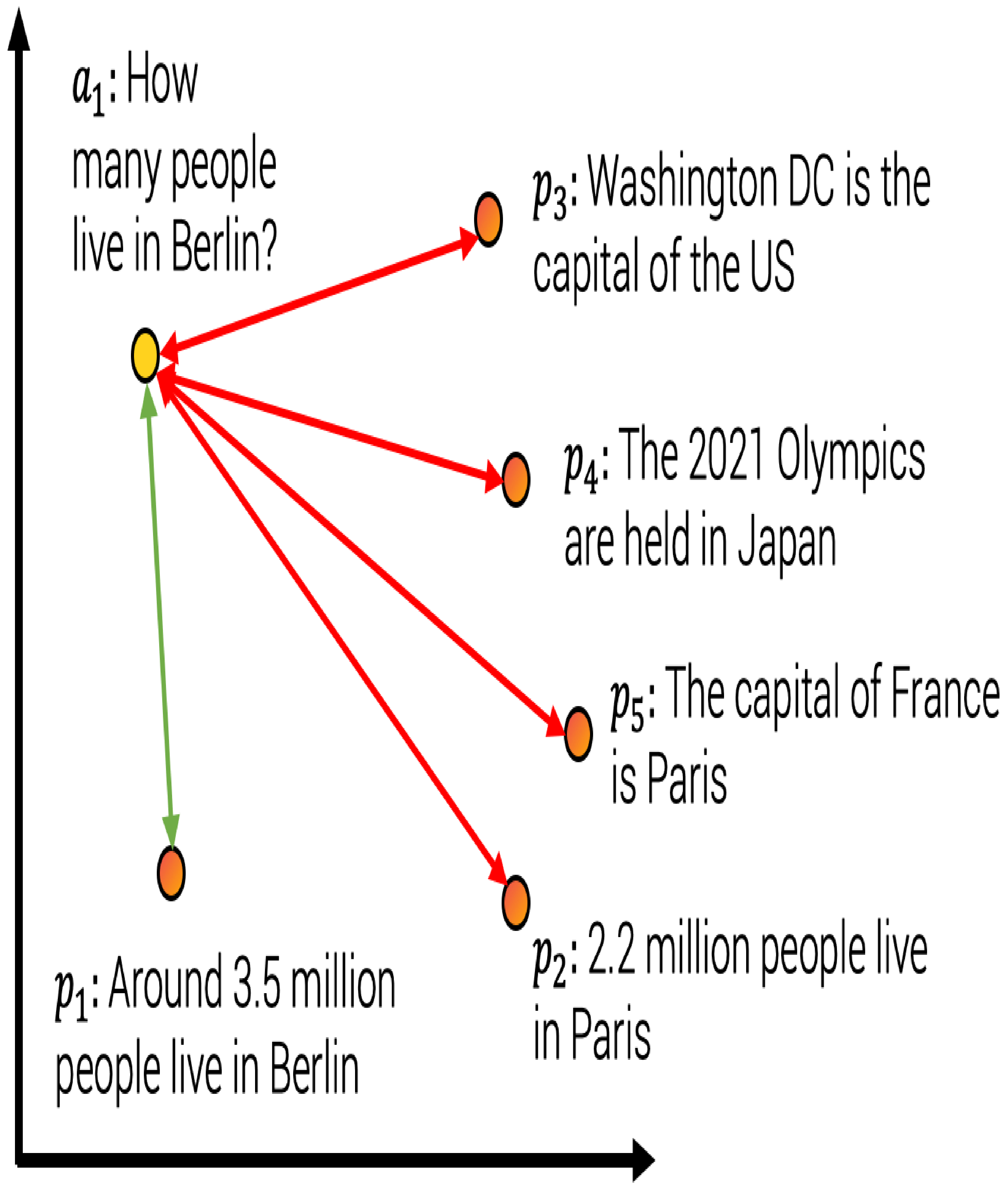
Many tasks require understanding full sentence meaning:

- Semantic search
- Paraphrase detection
- Question answering
- Dialogue systems

Solution:

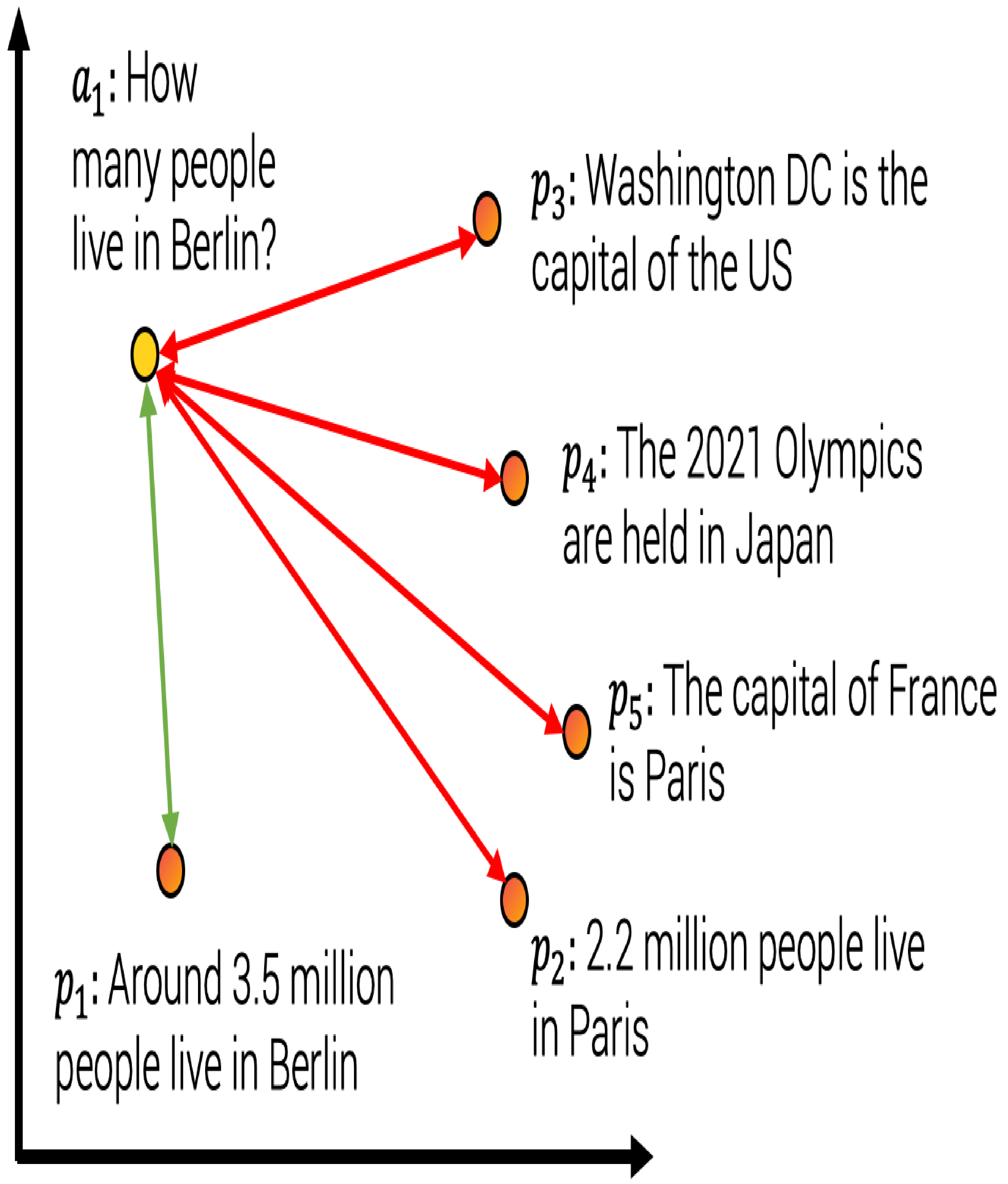
Use **Sentence Embeddings** — dense vector representations of entire sentences that preserve **meaning**, **structure**, and **context**.

Sentence Embeddings


a_1: How many people live in Berlin?
p_3: Washington DC is the capital of the US
p_4: The 2021 Olympics are held in Japan
p_5: The capital of France is Paris
p_1: Around 3.5 million people live in Berlin
p_2: 2.2 million people live in Paris

- **Sentence Embeddings** are dense vector representations that capture the **meaning of an entire sentence**, not just individual words.
- They convert a sentence into a **single fixed-length vector**
- This vector represents the **overall meaning, context, and structure** of the sentence
- Just like word embeddings represent words, sentence embeddings represent whole sentences.

Why It matter and Benefits


a_1: How many people live in Berlin?
p_3: Washington DC is the capital of the US
p_4: The 2021 Olympics are held in Japan
p_5: The capital of France is Paris
p_1: Around 3.5 million people live in Berlin
p_2: 2.2 million people live in Paris

- Capture **semantics beyond word-level**
- Preserve **word order and syntactic roles**
- Enable comparison between entire sentences or questions

Benefits:

Useful for tasks like:

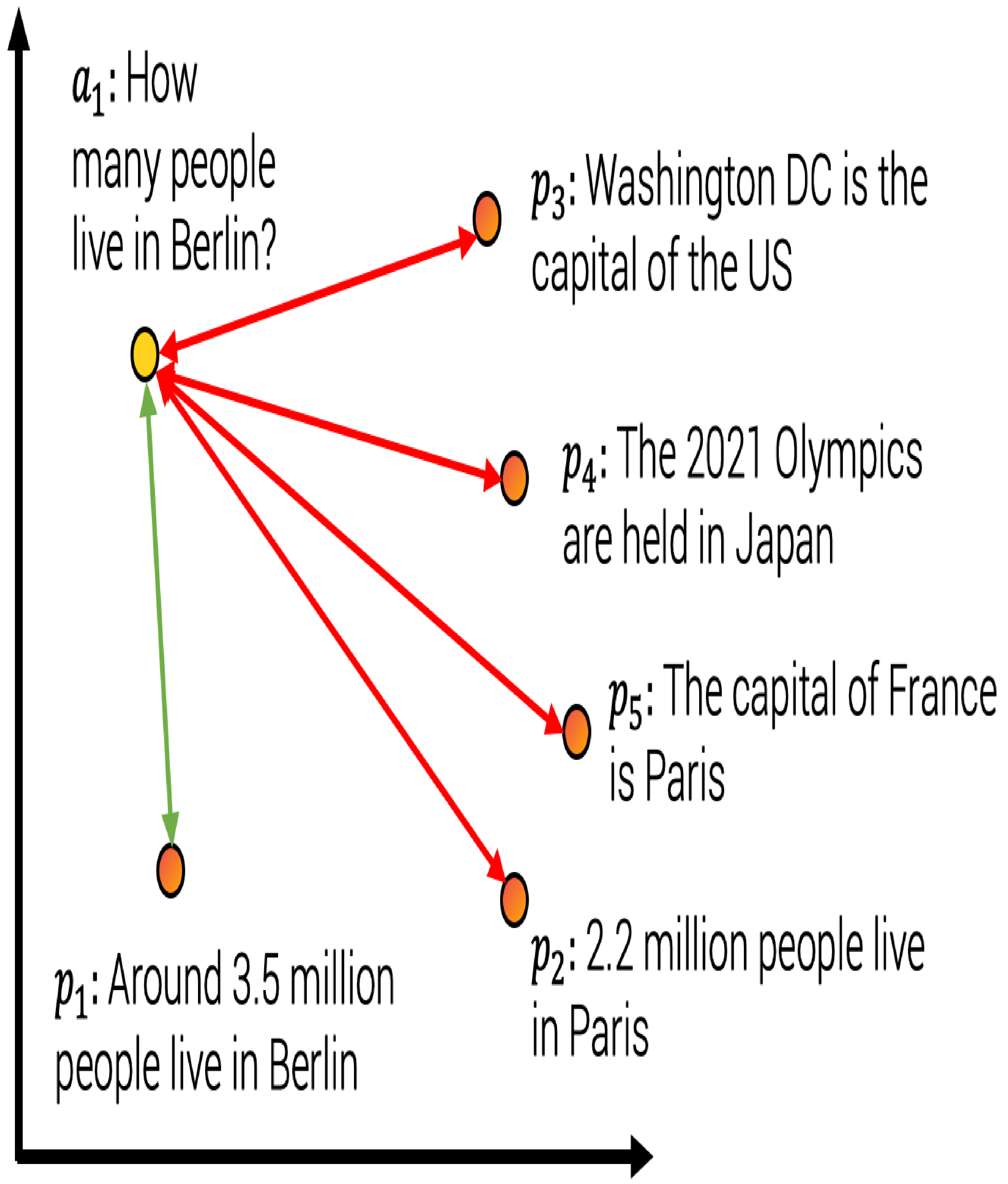
Semantic similarity
Paraphrase detection
Sentence clustering
Textual entailment

Can be compared using metrics like **cosine similarity**

Example:

Sentence A: “He is reading a book.”
Sentence B: “A book is being read by him.”
→ Sentence embeddings can recognize that both convey **the same meaning** despite different wordings.

Techniques for Sentence Embedding


a_1: How many people live in Berlin?
p_3: Washington DC is the capital of the US
p_4: The 2021 Olympics are held in Japan
p_5: The capital of France is Paris
p_1: Around 3.5 million people live in Berlin
p_2: 2.2 million people live in Paris

•To represent full sentence meaning as a vector, various **sentence embedding techniques** have been developed — ranging from simple to advanced.

1. Averaging Word Embeddings (Basic)

Takes pre-trained word vectors (like Word2Vec)

Computes the average across all words in the sentence

Simple and fast

Loses syntax, word order, and context

2. TF-IDF Weighted Averaging

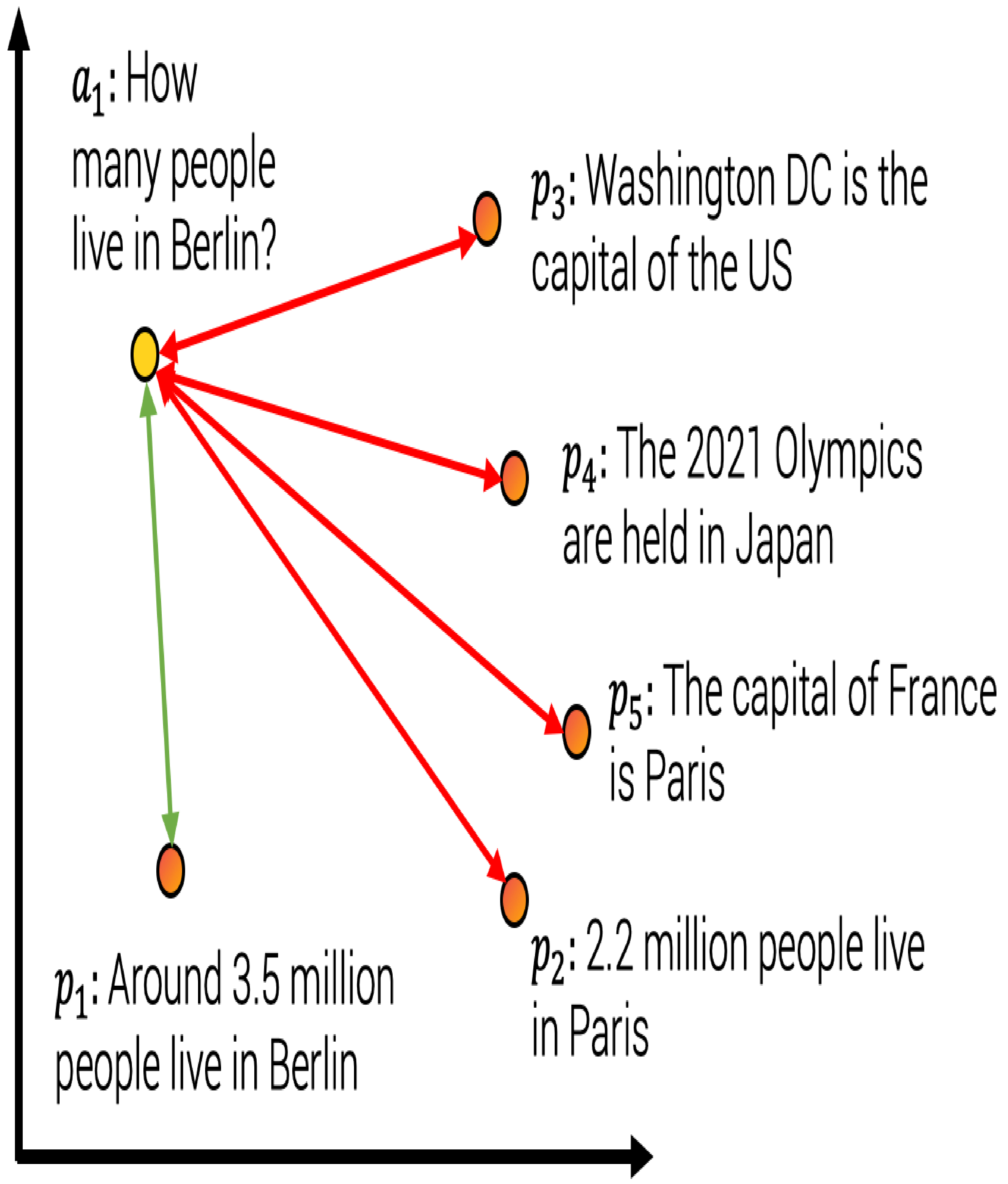
Weighs each word vector by its **importance** in the corpus (TF-IDF)

Reduces the influence of common or uninformative words

Improves over simple averaging

Still lacks true semantic depth or context

Techniques for Sentence Embedding


a_1: How many people live in Berlin?
p_3: Washington DC is the capital of the US
p_4: The 2021 Olympics are held in Japan
p_5: The capital of France is Paris
p_1: Around 3.5 million people live in Berlin
p_2: 2.2 million people live in Paris

3. Universal Sentence Encoder (USE)

Developed by Google

Produces fixed-length embeddings using **Transformer or DAN architectures**

Trained on a wide range of tasks for general-purpose sentence meaning

Strong performance across many tasks

Easy to use via TensorFlow Hub

4. InferSent

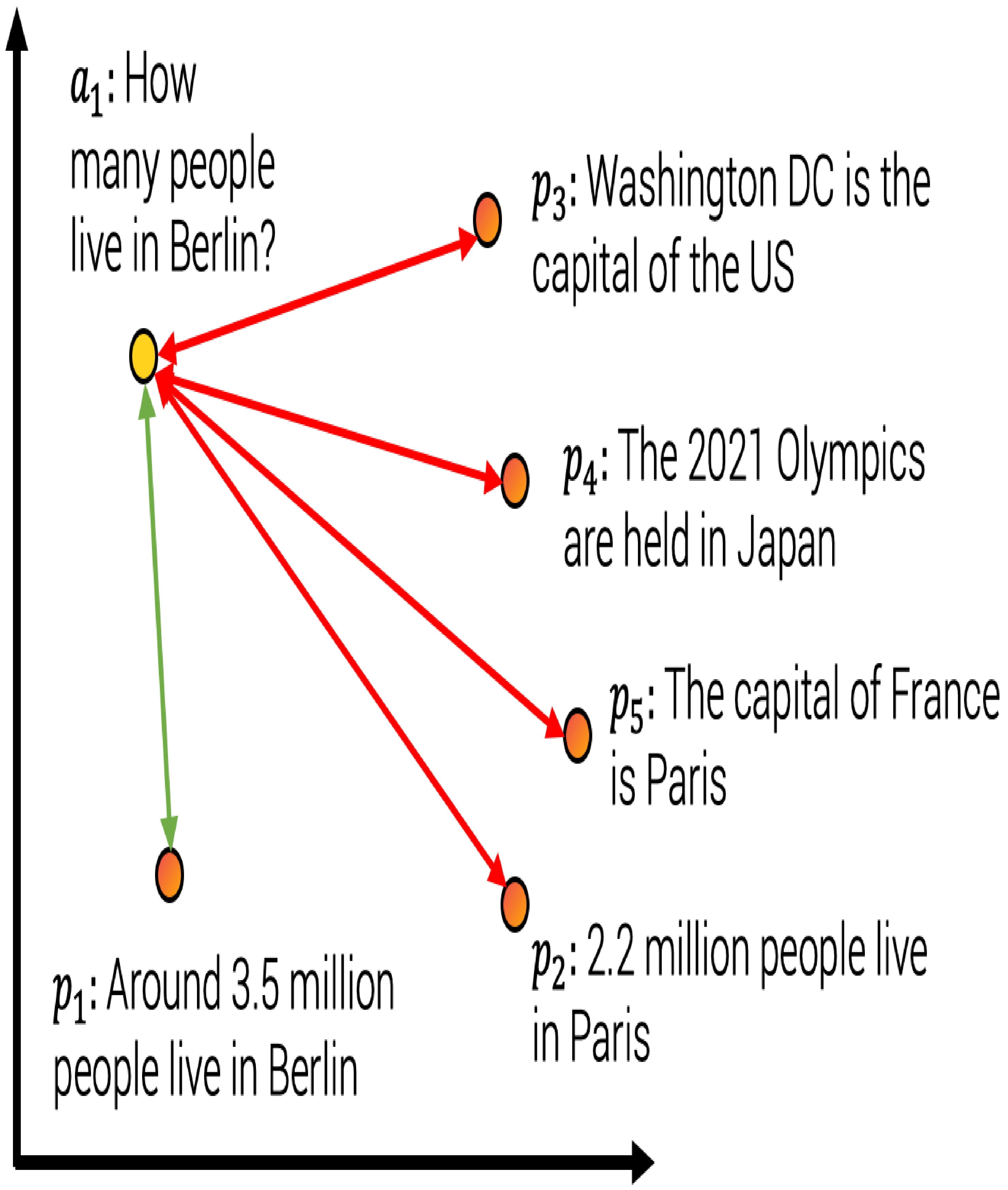
Developed by Facebook AI

Trained on **natural language inference (NLI)** data

Captures **semantic relationships and entailment**

Good for tasks requiring **logical sentence understanding**

Techniques for Sentence Embedding


a_1: How many people live in Berlin?
p_3: Washington DC is the capital of the US
p_4: The 2021 Olympics are held in Japan
p_5: The capital of France is Paris
p_1: Around 3.5 million people live in Berlin
p_2: 2.2 million people live in Paris

5. Sentence-BERT (SBERT)

Modified BERT architecture that uses **Siamese networks** to compare sentence pairs

Optimized for **semantic similarity and sentence matching**

State-of-the-art performance in semantic search, clustering, and paraphrase detection

Output vectors can be directly compared using **cosine similarity**

6. Doc2Vec

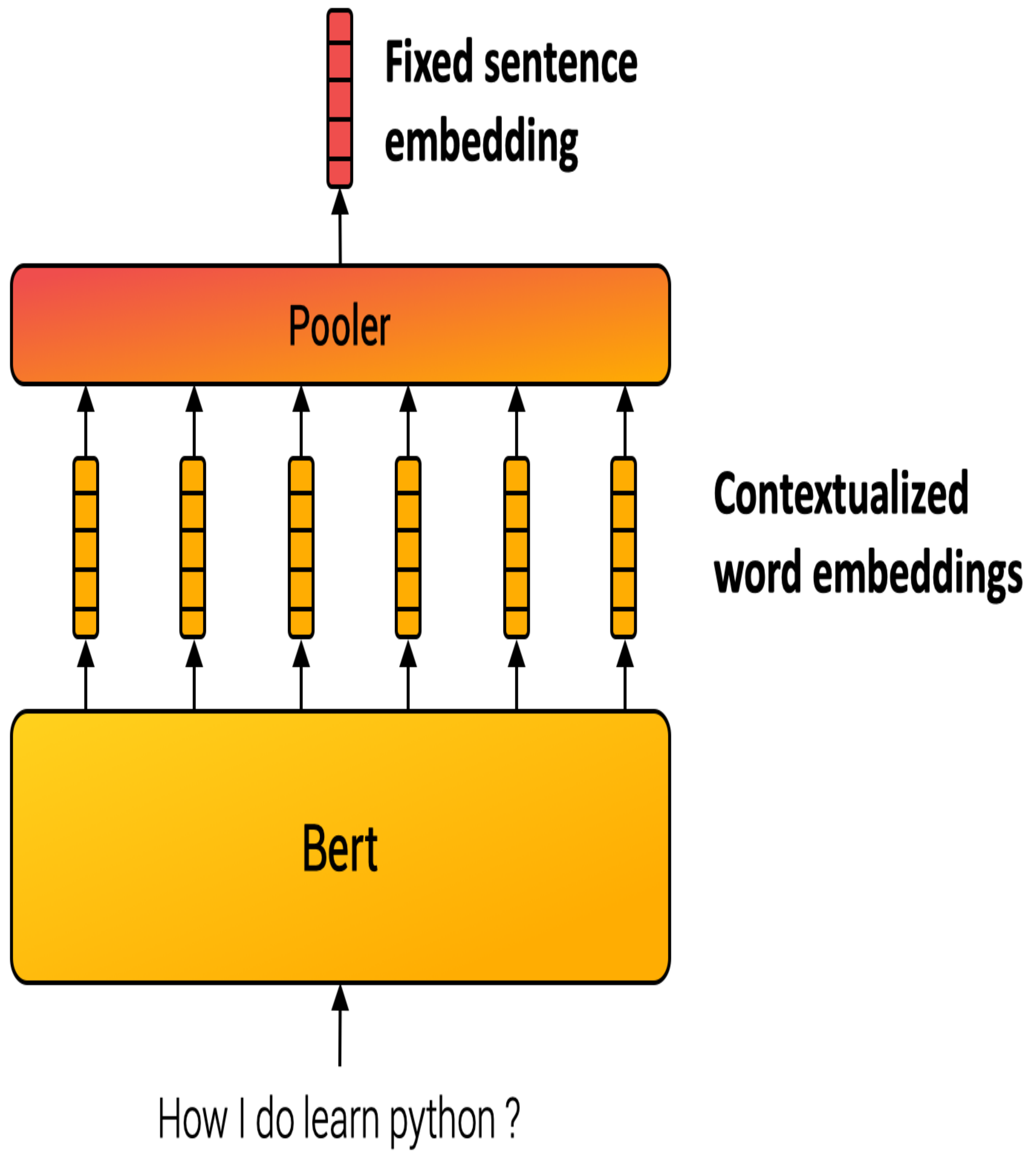
Extension of Word2Vec for longer texts

Learns a **vector for an entire document or paragraph**

Captures broader context

Less commonly used now due to newer Transformer models

Use Cases for Sentence Embeddings



Sentence embeddings power many advanced NLP applications by enabling machines to understand and compare **entire sentence meanings** efficiently.

1. Semantic Textual Similarity (STS)

•Measures how similar two sentences are in meaning
E.g., “I love pizza” ≈ “Pizza is my favorite food”
Used in duplicate question detection or content clustering

2. Question Answering

•Matches user questions with the most relevant answers
"How tall is the Eiffel Tower?" → links to factual sentence
Improves chatbots, search engines, and QA systems

4. Semantic Search

•Matches a query to documents based on **meaning**, not keywords
"Best laptop for students" → finds relevant reviews, even without exact terms
✅ Great for intelligent search engines

Combining Lexical and Semantic Methods

SEMANTIC
ANALYSIS

While **lexical** and **semantic** similarity methods each have strengths, combining them leads to **more accurate and robust** text analysis

- **Lexical methods** are fast and simple but miss deeper meaning
- **Semantic methods** understand context but are computationally heavier

Together, they balance **speed, precision, and depth**

How They Work Together:

Initial Filtering with Lexical Methods

Use methods like Jaccard or cosine similarity on TF-IDF or BoW

Quickly eliminate clearly irrelevant pairs

Deep Analysis with Semantic Models

Apply BERT, Sentence-BERT, or USE to the remaining candidates

Extract true semantic relationships

Benefits of Hybrid Approach

SEMANTIC ANALYSIS

- Reduces false positives/negatives
- Increases model confidence
- Adapts to varied language use (formal/informal, synonyms)

Example Workflow:

Step 1: Use TF-IDF + Cosine Similarity

→ Filter top 50 similar documents

Step 2: Use Sentence-BERT

→ Rank top 10 based on deep semantic similarity

- **Practical Applications:**

Task	Lexical + Semantic Combo Enhances...
Search engines	Precision and relevance
Duplicate detection	Catching exact and near duplicates
QA systems	Matching queries to answers effectively
Plagiarism detection	Finding reworded or paraphrased content

National Telecommunication Institute

Thanks