Understanding Word Embeddings: A Detailed Overview

What Are Word Embeddings?

Word embeddings are a technique for representing words as numerical **vectors**. These vectors capture the **meaning**, **context**, and **relationships** between words in a way that machines can interpret.

In these embeddings, words with similar meanings or that appear in similar contexts are positioned **closely together in multi-dimensional space**. This spatial relationship enables models to perform a variety of language tasks — such as finding similar words, understanding context, or solving analogies.

For example, the embedding space may reflect that the relationship between **"king"** and **"queen"** is similar to that between **"man"** and **"woman"** — something visible in the **direction and distance between their vectors**.

Why Are Word Embeddings Needed?

Machines can't process raw human language. NLP models require numerical input to learn and make predictions.

Word embeddings bridge the gap between text and machine learning by converting text into numerical form **while preserving meaning and context**. Without embeddings, models would treat "cat" and "dog" as completely unrelated character sequences. With embeddings, they can learn that these are both animals, pets, and semantically related — placing their vectors close to one another.

Where Are Word Embeddings Used?

Word embeddings are foundational to many NLP applications:

- **Text Classification** – e.g., spam detection, sentiment analysis.
- **Named Entity Recognition (NER)** – identifying names of people, places, or organizations.
- **Word Similarity & Analogy Tasks** – understanding relationships like *king is to queen as man is to woman*.
- **Question Answering & Recommendations** – computing semantic similarity between queries and documents.

- **Clustering & Semantic Search** – grouping or retrieving related content based on meaning.

How Are Word Embeddings Created?

Word embeddings are learned by training models on large text corpora such as Wikipedia or news articles. The general process includes:

1. **Preprocessing** – Cleaning text: tokenizing (splitting into words), removing punctuation and stopwords (e.g., "the", "is").
2. **Context Windowing** – Defining a word's context by looking at nearby words within a fixed window.
3. **Model Training** – Learning to predict a word from its context (or vice versa), adjusting vectors to minimize error.
4. **Vector Representation** – Each word ends up as a unique vector where similar meanings are encoded in proximity.

These vectors typically contain **hundreds of dimensions**, allowing them to capture complex language relationships and nuances.

Types of Word Embeddings

---

1. Frequency-Based Embeddings

These embeddings rely on how frequently words appear in documents and across corpora. The most common technique is:

- **TF-IDF (Term Frequency – Inverse Document Frequency)**
  Highlights words that appear **often in a specific document** but **not across all documents**.
  For instance, in a document about coffee, "espresso" or "latte" would rank higher than common words like "the".
  However, TF-IDF doesn't capture the **meaning** of words — only their **statistical presence**.

---

2. Prediction-Based Embeddings

These embeddings learn from the **context** in which words appear. They are trained to **predict a word from its surrounding words**, or vice versa.

Example:
In the sentences:

- “The dog is wagging its tail.”
- “The dog is barking loudly.”
  The model sees that "dog" often co-occurs with "wag", "tail", "bark", and "pet". As a result, their vectors are close together — not because of frequency, but **contextual relationship**.

Prediction-based embeddings also handle **polysemy** — where words have multiple meanings.
For example, the model learns different vectors for **“bank”** as a financial institution vs a riverbank based on surrounding words.

Popular Word Embedding Models

Word2Vec

Developed by Google, Word2Vec is one of the most widely used predictive embedding models. It has two training strategies:

- **CBOW (Continuous Bag of Words)** – Predicts a word based on its context.
  Example: Given “The cat sat on the mat,” CBOW might use ["The", "cat", "on", "the"] to predict "sat".
- **Skip-Gram** – Does the reverse: predicts surrounding words from a given word.
  Example: Using "sat", predict ["The", "cat", "on", "the"].

This model helps place similar words (like "cat" and "mat") close together in vector space.

GloVe

GloVe (Global Vectors for Word Representation) uses **global word co-occurrence** statistics instead of just local context.

It examines how frequently words appear together across the entire corpus, capturing broader semantic relationships.

Example from a toy corpus:

- "Ice cream is cold"

- "The weather is cold"
- "Ice melts in heat"
- "Sun brings heat"

GloVe observes that:

- **cold** often appears with **ice** and **weather**
- **heat** often appears with **sun** and **melts**

Thus, it infers associations like **cold ↔ ice**, **heat ↔ sun**.

Key Difference:

- **Word2Vec**: Learns from **local context** (neighboring words)
- **GloVe**: Learns from **global statistics** (overall co-occurrence)

FastText

FastText, developed by Facebook, enhances Word2Vec by considering **subword information** (character n-grams).

Instead of treating “playing” as one word, it breaks it into smaller parts: “pla”, “lay”, “ing”, etc. This helps it:

- Handle **misspelled or rare words**
- Understand **word structure** and **morphology**

Example: Even if “playz” is a new or misspelled word, FastText can infer meaning from known subwords.

Contextual Embeddings: A Step Beyond

Traditional embeddings assign **a single, fixed vector** to each word, regardless of how it's used. But language is contextual — and meanings change based on usage.

Contextual embeddings, introduced by transformer models like BERT, generate **different vectors** for the same word depending on its sentence.

Example:

- “I went to the **bank** to deposit money.”
- “We sat on the **bank** of the river.”
  In a traditional model, “bank” would have one fixed vector.

In contextual models, the vector changes — capturing its financial or geographic sense.

This makes contextual embeddings especially powerful for tasks involving **ambiguity, sentence meaning, and real-world language understanding**.

TYPES OF EMBEDDINGS

---

1. No Contextual Understanding (Pre-contextual Embeddings)

Fixed embeddings for words — same vector regardless of context. Based on frequency or shallow prediction.

Methods:

- **Latent Semantic Analysis (LSA)** – Finds hidden topics in documents by looking at word patterns.
- **Latent Dirichlet Allocation (LDA)** – Automatically finds topics in a collection of documents.
- **Word2Vec** – Learns word meanings by predicting words that appear near each other. *(CBOW & Skip-gram)*
- **GloVe** – Creates word vectors by counting how often words appear together across the whole text.
- **FastText** – Breaks words into smaller parts to better handle rare or unknown words.

---

2. Contextual Understanding (Medium-Scale Contextual Models)

Generates different embeddings based on word context. Typically based on transformer or LSTM architectures.

Methods:

- **ELMo** – Learns word meaning by looking at both the left and right context using bidirectional LSTMs.
- **OpenAI GPT (GPT-1)** – Predicts the next word based on left-to-right context using transformers.
- **BERT** – Understands context by looking at both directions in a sentence simultaneously.

- **RoBERTa** – Improves BERT by training on more data without next-sentence prediction.
- **ALBERT** – A lightweight and memory-efficient version of BERT.
- **XLNet** – Enhances BERT using a permutation-based training method.
- **DistilBERT** – A smaller, faster version of BERT with comparable performance.
- **Sentence-BERT (SBERT)** – Modified BERT for embedding whole sentences.
- **ELECTRA** – Trains efficiently by detecting replaced (fake) words instead of masking.

---

3. Large Foundation Models (LLMs) with Contextual Embeddings and Advanced Capabilities

Massive transformer-based models with billions of parameters. Designed for reasoning, multitask learning, generation, and deep contextual understanding.

Methods:

- **T5 (Text-to-Text Transfer Transformer)** – Treats all NLP tasks as text-to-text problems to learn task-aware embeddings.
- **GPT-2 embeddings (fine-tuned use)** – Extracts rich embeddings from intermediate layers of a generative model.
- **GPT-3 embeddings** – Deep contextual vectors from OpenAI's large-scale autoregressive model.
- **CLIP (OpenAI)** – Aligns text and images in a shared vector space using contrastive learning.
- **PaLM (Google)** – A multilingual LLM with strong generalization and reasoning capabilities.
- **Chinchilla (DeepMind)** – Optimized training for better performance per parameter.
- **LLaMA (Meta)** – Open-access LLMs designed for efficient training and high-quality embeddings.
- **GPT-4 embeddings** – Advanced embeddings from OpenAI's flagship model with strong task generalization.
- **Mistral models** – Open-source models that are compact and performant for various downstream tasks.

- **Gemini (Google DeepMind)** – Multimodal embeddings trained across text, code, and images.
- **Claude (Anthropic)** – Safety-aligned embeddings from models designed for aligned reasoning.
- **GPT-4o embeddings** – Multimodal embeddings from GPT-4o trained on text, images, and audio.

RESOURCE : https://www.youtube.com/watch?v=wgfSDrqYMJ4

https://www.deeplearning.ai/short-courses/building-applications-vector-databases/