From Words to Vectors: Converting Sentences to High-Dimensional Representations

Department of Computer Science & IT

Abstract

This document explains how sentences are transformed into high-dimensional vectors using modern natural language processing techniques. We explore the step-by-step process from tokenization to contextual embeddings, with visual examples and mathematical explanations to help students understand this fundamental concept in NLP.

Introduction

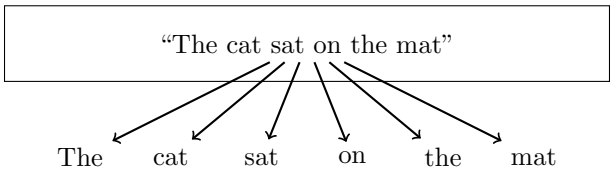
In natural language processing, we need to convert human language into a format that computers can understand and process. The solution is to represent words and sentences as **vectors** in a high-dimensional space. This transformation allows us to perform mathematical operations on language, enabling tasks like sentiment analysis, machine translation, and search.

1 The Transformation Process

The conversion of a sentence to a vector involves several key steps:

1.1 Step 1: Tokenization

The sentence is broken down into smaller units called tokens (words, subwords, or characters).



1.2 Step 2: Word Embeddings

Each token is mapped to a dense vector using a pre-trained embedding model. These vectors capture semantic meaning:

- Similar words have similar vectors
- Relationships can be expressed through vector arithmetic

"cat"	[0.7, -0.2, 0.4, ..., 0.1]

1.3 How Numerical Values Are Obtained

The numerical values in word vectors (e.g., $[0.7, -0.2, 0.4, \ldots, 0.1]$) are **learned parameters** from neural network training. Here's how they're generated:

1. **Training Objective**: Models like Word2Vec learn by predicting context words. For a target word w_t and context window size c, the model maximizes:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c}^{j\neq 0}\log p(w_{t+j}|w_t)$$

where $p(w_O|w_I) = \frac{\exp(\mathbf{v}_{w_O}^\top \mathbf{v}_{w_I})}{\sum_{w=1}^{V}\exp(\mathbf{v}_w^\top \mathbf{v}_{w_I})}$ uses dot-product similarity.

2. **Matrix Factorization**: Methods like GloVe solve:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

   where X_{ij} is word co-occurrence frequency.

3. **Neural Network Weights**: The values come from weight matrices in shallow networks:
   - Input layer → Hidden layer weights = *word vectors*
   - Hidden layer → Output layer weights = *context vectors*

4. **Geometric Interpretation**:
   - Each dimension encodes latent features (e.g., animacy, tense)
   - Similar words cluster geometrically: $\|$"cat" − "dog"$\|$ is small
   - Analogies hold linearly: "king" − "man" + "woman" ≈ "queen"

Why these specific numbers?
The values emerge from optimization on massive corpora (e.g., Wikipedia). There's no predefined meaning for individual dimensions – they collectively encode distributed representations through training dynamics.

1.4 Step 3: Contextual Embeddings

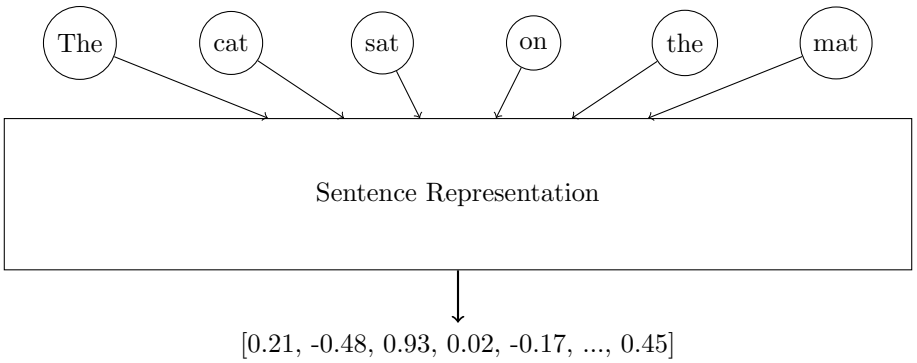
Modern models (like BERT) create contextual embeddings where the vector for a word changes based on its context:

"I accessed my **bank** account online" → [0.2, 0.8, -0.3, ..., 0.4]

"We fished by the river **bank**" ⟶ [-0.1, 0.2, 0.7, ..., -0.5]

1.5 Step 4: Sentence Representation

The individual word vectors are combined to form a single vector representing the entire sentence:



2 Mathematical Representation

A sentence S is represented as a vector in $\mathbb{R}^n$ where n is typically 300-1024 dimensions:

$$\vec{s} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

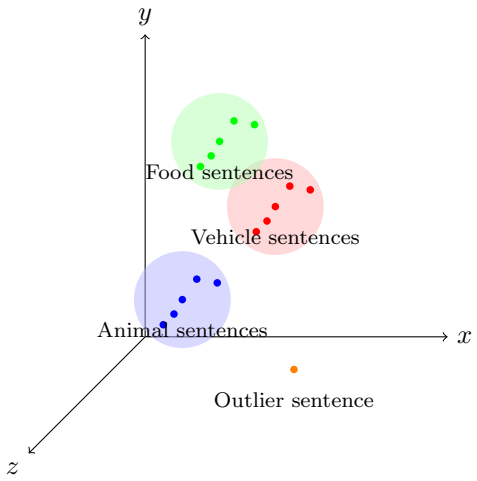
2.1 Similarity Measurement

The similarity between two sentences is calculated using cosine similarity:

$$\text{cosine similarity} = \frac{\vec{s_1} \cdot \vec{s_2}}{\|\vec{s_1}\|\|\vec{s_2}\|}$$

2.2 Visualizing Semantic Space

In high-dimensional space, similar sentences cluster together:



3 Why High Dimensions?

High-dimensional spaces provide several advantages:

- **Expressiveness**: Can capture subtle semantic differences
- **Capacity**: Can represent millions of words without collisions
- **Geometry**: Enables linear relationships (e.g., king - man + woman queen)
- **Manifold Learning**: Sentences lie on complex geometric structures

4 Common Embedding Models

Model	Dimensions	Type	Features
Word2Vec	300	Static	Efficient, captures semantic relationships
GloVe	300	Static	Global co-occurrence statistics
FastText	300	Static	Handles OOV words with subword info
BERT	768	Contextual	Bidirectional, transformer architecture
RoBERTa	1024	Contextual	Optimized BERT with more data
Sentence-BERT	768	Sentence-level	Fine-tuned for sentence similarity

5 Example with Sentence-BERT

Here's how to generate sentence embeddings in Python:

```python
from sentence_transformers import SentenceTransformer

# Load pre-trained model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Sentences to encode
sentences = [
    "The cat sat on the mat",
    "A feline rested on the rug",
    "Cars drive on the highway"
]

# Generate embeddings
embeddings = model.encode(sentences)

# Output will be 3 vectors of 384 dimensions
print(embeddings.shape)  # (3, 384)

# Calculate similarity between first and second sentence
from sklearn.metrics.pairwise import cosine_similarity
sim = cosine_similarity([embeddings[0]], [embeddings[1]])
```

```
print(f"Similarity: {sim[0][0]:.2f}")  # Should be high (>0.8)

# Calculate similarity between first and third sentence
sim = cosine_similarity([embeddings[0]], [embeddings[2]])
print(f"Similarity: {sim[0][0]:.2f}")  # Should be low (<0.3)
```

Listing 1: Generating Sentence Embeddings

Conclusion

Converting sentences to high-dimensional vectors is fundamental to modern NLP. This process:

- Transforms text into a mathematical representation
- Captures semantic meaning and relationships
- Enables computational analysis of language
- Powers applications like search, translation, and chatbots

Understanding this transformation is essential for working with advanced language models and developing NLP applications.