Distributed Representations Beyond Words and Characters

Sentence/Paragraph Embeddings in NLP Pipeline

Start Presentation

Advanced NLP • Text Representation

The NLP Pipeline: Text Representation Stage

Traditional Approach

- Tokenization → Words/Characters
- Word Embeddings (Word2Vec, GloVe)
- **Limitations:** Fixed vocabulary, no context

Modern Approach

- Contextual Understanding
- Variable-length sequences
- Semantic composition
- Beyond individual tokens

Key Challenge: How do we represent entire sentences and paragraphs as dense vectors while preserving semantic meaning?

From Words to Sentences: The Challenge

The Compositional Problem

```
# Word-level embeddings
word_vectors = {
    "cat": [0.2, 0.8, 0.1],
    "sat": [0.5, 0.3, 0.7],
}

# Simple averaging loses context
sentence = "The cat sat on the mat"
simple_avg = average(word_vectors)
```

Problem: Simple averaging loses word order, syntax, and semantic relationships

Issues with Simple Approaches

Simple Averaging Issues

- Loses word order
- Ignores syntax
- No semantic composition

Bag of Words Problem

- "Dog bites man" ≈ "Man bites dog"
- Same vectors, different meaning
- Context-free representation

What We Need

- Preserve semantics
- Consider context
- Maintain relationships

Solution: We need methods that can capture semantic meaning while preserving contextual relationships

What are Sentence Embeddings?

Definition

Dense vector representations that capture semantic meaning of entire sentences in fixed-dimensional space.

Key Properties

- Fixed dimensionality (256, 512, 768)
- Semantic similarity preserved
- Context-aware representations
- Task-agnostic or task-specific

Mathematical Representation

```
Sentence: "The weather is beautiful today"
          ↓
Encoder Function f(·)
          ↓
Vector: [0.23, -0.45, 0.67, ...]
        ∈ ℝ^d (d = embedding dimension)
```

Semantic Similarity:
Similar sentences → Similar vectors
$\text{cosine}(v_1, v_2) \approx 1$ if sentences are similar

Methods for Creating Sentence Embeddings - Part 1

1. Aggregation Methods

- Simple Averaging
- Weighted Averaging (TF-IDF)
- Max/Min Pooling

```
# Weighted average example
def weighted_embedding(words, weights):
    return sum(w * embed(word)
               for word, w in zip(words, weights))
```

2. Sequential Models

- RNNs/LSTMs
- Last hidden state as embedding
- Bidirectional processing

```
# LSTM sentence embedding
lstm = LSTM(hidden_size=256)
hidden_states = lstm(word_embeddings)
sentence_embedding = hidden_states[-1]
```

Methods for Creating Sentence Embeddings - Part 2

3. Transformer-based

- Self-attention mechanisms
- BERT, RoBERTa ([CLS] token)
- Sentence-BERT (fine-tuned)

```
# BERT sentence embedding
sentence = "NLP is amazing"
inputs = tokenizer(sentence, return_tensors="pt")
outputs = model(**inputs)
embedding = outputs.last_hidden_state[:, 0, :]
```

4. Specialized Architectures

- Universal Sentence Encoder
- InferSent
- Quick Thoughts
- SimCSE

```
# Universal Sentence Encoder
import tensorflow_hub as hub
encoder = hub.load("universal-sentence-encoder")
embeddings = encoder(["Hello world"])
```

Popular Models: Universal Sentence Encoder

Architecture & Features

- **Developer:** Google Research
- **Architecture:** Transformer + Deep Averaging Network
- **Dimensions:** 512
- **Languages:** Multilingual support
- **Training:** Multiple tasks (SNLI, STS, etc.)

Strengths & Use Cases

- **Fast inference:** Optimized for production
- **General purpose:** Works across domains
- **Easy integration:** TensorFlow Hub
- **Good baseline:** Solid performance
- **Multilingual:** 16+ languages

Popular Models: Sentence-BERT (SBERT)

Architecture & Features

- **Developer:** UKP Lab
- **Architecture:** BERT with Siamese network
- **Dimensions:** 768 (configurable)
- **Training:** Optimized for similarity tasks
- **Fine-tuning:** Task-specific adaptation

Key Advantages

- **Speed:** 65x faster than BERT pairs
- **Quality:** State-of-the-art similarity
- **Flexibility:** Multiple model sizes
- **Community:** Extensive model zoo
- **Production-ready:** Optimized inference

Popular Models: InferSent

Architecture & Features

- **Developer:** Facebook Research
- **Architecture:** BiLSTM with max pooling
- **Dimensions:** 4096
- **Training:** Stanford Natural Language Inference
- **Focus:** Transfer learning

Performance Characteristics

- **Transfer learning:** Good downstream performance
- **Interpretability:** Attention mechanisms
- **Robustness:** Handles various text types
- **Research focus:** Academic applications

Model Performance Comparison

Benchmark Results

Model	Dimensions	Speed	STS Score	Best Use Case
USE	512	Fast	0.78	General purpose
SBERT	768	Very Fast	0.85	Similarity search
InferSent	4096	Moderate	0.75	Transfer learning
SimCSE	768	Fast	0.84	Contrastive learning

Speed Comparison:
SBERT: 65x faster than BERT pairs
USE: Optimized for production
InferSent: Moderate inference time

Quality Metrics:
STS = Semantic Textual Similarity
Higher scores = better similarity
Benchmark: STS-B dataset

Paragraph Embeddings - Introduction

What are Paragraph Embeddings?

Dense vector representations for entire documents or paragraphs that capture semantic meaning beyond individual sentences.

Key Characteristics

- **Variable Length Handling:** Process documents of any size
- **Hierarchical Structure:** Capture both local and global context
- **Semantic Coherence:** Maintain meaning across long text spans
- **Document-Level Features:** Beyond sentence-level understanding

Challenge: How do we maintain semantic coherence across long documents while capturing both local details and global structure?

Doc2Vec Approaches

PV-DM (Distributed Memory)

```
# Predict word given context + paragraph vector
P(word | context_words, paragraph_id)
```

Characteristics:

- Uses paragraph vector as additional context
- Similar to Word2Vec CBOW
- Better for smaller datasets
- Preserves word order information

PV-DBOW (Distributed Bag of Words)

```
# Predict words given paragraph vector
P(words | paragraph_id)
```

Characteristics:

- Ignores word order in context
- Similar to Word2Vec Skip-gram
- Faster training, good performance
- More efficient for large corpora

Best Practice: Combine both PV-DM and PV-DBOW for optimal results

Hierarchical Attention Networks

Two-Level Attention Mechanism

Word-Level Attention

- Identifies important words within sentences
- Creates sentence representations
- Preserves local context

```
# Word attention example
word_attention = attention(words)
sentence_vectors = aggregate(word_attention)
```

Sentence-Level Attention

- Identifies important sentences in documents
- Creates document representations
- Captures global structure

```
# Sentence attention example
sentence_attention = attention(sentence_vectors)
document_vector = aggregate(sentence_attention)
```

Complete Process

1. **Words → Sentences:** Word-level attention aggregates words into sentence vectors
2. **Sentences → Document:** Sentence-level attention aggregates sentences into document vector
3. **Hierarchical Understanding:** Captures both local and global semantic information

Modern Transformer-Based Approaches

Long-Sequence Transformers

Longformer

- **Sparse attention patterns**
- Linear complexity O(n)
- Sliding window + global attention
- Up to 4,096 tokens

BigBird

- **Random + global + local attention**
- Theoretical guarantees
- Efficient for long documents
- Graph-based attention

Hierarchical BERT Approaches

Document Chunking

- Process documents in overlapping chunks
- Combine chunk representations
- Maintain global context across chunks

Sentence-Level Processing

- Encode sentences individually
- Apply document-level transformer
- Preserve hierarchical structure

Trade-off: Computational efficiency vs. representation quality vs. maximum sequence length

Applications: Information Retrieval & Search

Semantic Search Systems

- **Query understanding:** Convert queries to embeddings
- **Document ranking:** Similarity-based retrieval
- **Cross-lingual search:** Multilingual embeddings
- **Personalization:** User-specific representations

Question-Answering

- **Passage retrieval:** Find relevant text segments
- **Answer extraction:** Locate specific information
- **Context understanding:** Maintain conversation state

Implementation Example

```
from sentence_transformers import SentenceTransformer
import numpy as np

model = SentenceTransformer('all-MiniLM-L6-v2')

corpus = [
    "ML is a subset of AI",
    "Deep learning uses neural networks",
    "NLP handles text processing",
    "Computer vision processes images"
]
# Encode corpus
corpus_embeddings = model.encode(corpus)
# Search query
query = "What is artificial intelligence?"
query_embedding = model.encode([query])
# Find similarities
similarities = np.dot(query_embedding, corpus_embeddings.T
top_idx = np.argmax(similarities)
print(f"Best match: {corpus[top_idx]}")
```

Applications: Text Classification & Analysis

Classification Tasks

Sentiment Analysis

- Movie reviews, product feedback
- Social media monitoring
- Customer satisfaction

Topic Classification

- News categorization
- Email routing
- Content organization

Intent Detection

- Chatbot understanding
- Voice assistants
- Customer service automation

Clustering & Similarity

Document Clustering

- Automatic topic discovery
- Content organization
- Research paper grouping

Plagiarism Detection

- Academic integrity
- Content originality
- Copyright protection

Recommendation Systems

- Content-based filtering
- Similar article suggestions
- Personalized feeds

Technical Challenges

Computational Complexity

Attention Complexity

- Quadratic complexity $O(n^2)$
- Memory requirements scale rapidly
- GPU memory limitations

Training Resources

- Large datasets required
- Computational cost
- Energy consumption

Representation Quality

Information Loss

- Fixed-size bottleneck
- Loss of fine-grained details
- Compression artifacts

Domain Adaptation

- General vs. domain-specific models
- Transfer learning challenges
- Out-of-domain performance

Evaluation and Metrics

Intrinsic Evaluation

Semantic Textual Similarity (STS)

- Pearson correlation with human judgments
- STS-B benchmark dataset
- Cross-lingual STS tasks

Clustering Metrics

- Silhouette score
- Adjusted Rand Index
- Normalized Mutual Information

Extrinsic Evaluation

Downstream Task Performance

- Classification accuracy
- Information retrieval metrics
- Question-answering performance

Practical Metrics

- Inference speed
- Memory usage
- Scalability measures

Key Trade-off: Expressiveness vs. Efficiency vs. Generalizability