8.Word Embeddings in Natural Language Processing

Definition

Word embedding is a technique in Natural Language Processing (NLP) used for representing words as real-valued vectors for text analysis. These vectors encode the semantic meaning of words such that words closer in the vector space are expected to be similar in meaning.

Core Concept

Word embeddings transform words into numerical vectors that capture semantic relationships. For example:

- **Similar words** like "happy" and "excited" will have vectors close to each other in the vector space
- **Opposite words** like "happy" and "angry" will have vectors far apart from each other

Distance Representation

The semantic similarity between words can be measured using distance metrics in vector space:

$$d(\vec{w}_1, \vec{w}_2) = \sqrt{\sum_{i=1}^{n} (w_{1i} - w_{2i})^2}$$

Where:

- $d(\vec{w}_1, \vec{w}_2)$ = Euclidean distance between two word vectors
- $\vec{w}_1, \vec{w}_2$ = word vectors in n-dimensional space
- w_{1i}, w_{2i} = individual components of the word vectors
- n = dimensionality of the word embedding space

Interpretation: Smaller distance indicates higher semantic similarity between words.

Types of Word Embedding Techniques

Word embedding techniques can be broadly classified into two categories:

1. Count/Frequency-Based Methods

These methods rely on statistical properties of word occurrences in text:

a) One-Hot Encoding

Represents each word as a binary vector with dimensionality equal to vocabulary size:

$$\vec{w}_i = [0, 0, ..., 1, ..., 0]$$

Where:

- $\vec{w}_i$ = one-hot encoded vector for word i
- The vector has length $|V|$ (vocabulary size)
- Only the i-th position is 1, all others are 0

Limitations:

- High dimensionality (sparse vectors)
- No semantic relationship captured
- Memory inefficient for large vocabularies

b) Bag of Words (BoW)

Represents documents as vectors based on word frequency:

$$\vec{d} = [f_1, f_2, ..., f_{|V|}]$$

Where:

- $\vec{d}$ = document vector
- f_i = frequency of word i in the document
- $|V|$ = total vocabulary size

Limitations:

- Loses word order information
- High dimensionality
- No consideration of word importance

c) TF-IDF (Term Frequency-Inverse Document Frequency)

Weights words based on their frequency in a document relative to their frequency across all documents:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

Where:

Term Frequency:

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

- $f_{t,d}$ = frequency of term t in document d
- $\sum_{t' \in d} f_{t',d}$ = total number of terms in document d

Inverse Document Frequency (Standard Smoothed Version):
$\text{IDF}(t) = \log_e \left(\frac{N+1}{n_t+1} \right) + 1$

- $\log_e$ = natural logarithm (logarithm to base e)
- N = total number of documents (or sentences) in the corpus
- n_t = number of documents (or sentences) containing term t
- $+1$ in numerator and denominator = smoothing to prevent division by zero
- Final $+1$ = ensures all IDF values are positive (≥ 1)

Interpretation:

- TF measures how frequently a term appears in a document
- IDF reduces the weight of commonly occurring words
- Higher TF-IDF score indicates greater importance of the term to that specific document

Limitations:

- Still results in sparse, high-dimensional vectors
- Does not capture semantic relationships between words
- Computational complexity increases with vocabulary size

2. Deep Learning-Based Methods (Predictive Models)

These methods use neural networks to learn dense vector representations that capture semantic relationships.

Word2Vec

Word2Vec is a group of neural network models that learn word embeddings by predicting word contexts. It produces dense vectors that capture semantic and syntactic relationships.

Objective Function:

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

Where:

- $J(\theta)$ = objective function to maximize
- T = total number of words in the corpus
- c = context window size
- w_t = target word at position t
- w_{t+j} = context word at position $t+j$
- θ = model parameters
- $P(w_{t+j}|w_t;\theta)$ = probability of context word given target word

Word2Vec has two main architectures:

a) Continuous Bag of Words (CBOW)

CBOW predicts the target word from surrounding context words:

$$P(w_t|w_{t-c},...,w_{t-1},w_{t+1},...,w_{t+c}) = \frac{\exp(\vec{v'}_{w_t}^T\cdot\vec{h})}{\sum_{w\in V}\exp(\vec{v'}_w^T\cdot\vec{h})}$$

Where:

Hidden Layer:

$$\vec{h} = \frac{1}{2c}\sum_{-c\leq j\leq c, j\neq 0}\vec{v}_{w_{t+j}}$$

- $\vec{h}$ = hidden layer representation (average of context word vectors)
- $\vec{v}_{w_{t+j}}$ = input vector for context word at position $t+j$
- c = context window size
- $\vec{v'}_{w_t}$ = output vector for target word w_t
- V = vocabulary

Interpretation:

- CBOW averages the context word vectors to predict the center word
- Faster to train and works well with smaller datasets
- Better for frequent words

b) Skip-gram

Skip-gram predicts context words from a target word:

$$P(w_{t+j}|w_t) = \frac{\exp(\vec{v'}_{w_{t+j}}^T \cdot \vec{v}_{w_t})}{\sum_{w \in V} \exp(\vec{v'}_w^T \cdot \vec{v}_{w_t})}$$

Where:

- w_t = target (center) word
- w_{t+j} = context word at position $t + j$
- $\vec{v}_{w_t}$ = input vector for target word
- $\vec{v'}_{w_{t+j}}$ = output vector for context word
- V = vocabulary

Interpretation:

- Skip-gram uses one word to predict multiple context words
- Works better with smaller amounts of training data
- Better representation for rare words
- Generally produces more accurate vectors but slower to train

Optimization Techniques for Word2Vec

Negative Sampling

To improve computational efficiency, negative sampling approximates the softmax:

$$\log \sigma(\vec{v'}_{w_O}^T \cdot \vec{v}_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-\vec{v'}_{w_i}^T \cdot \vec{v}_{w_I})]$$

Where:

- $\sigma(x) = \frac{1}{1+e^{-x}}$ = sigmoid function
- w_O = observed context word (positive sample)
- w_I = input word

- k = number of negative samples
- $P_n(w)$ = noise distribution for sampling negative examples
- w_i = randomly sampled "negative" words

Interpretation: Instead of computing probabilities over the entire vocabulary, we only update weights for a small number of negative examples along with the positive example.

Advantages of Deep Learning-Based Embeddings

1. **Dense Representations**: Low-dimensional vectors (typically 50-300 dimensions) instead of vocabulary-sized sparse vectors
2. **Semantic Relationships**: Captures meaning and relationships between words
3. **Arithmetic Properties**: Vector operations reflect semantic relationships: $\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$
4. **Transfer Learning**: Pre-trained models (like Google's Word2Vec trained on Google News corpus, ~1.5 GB) can be used across different tasks
5. **Better Generalization**: Handles unseen word combinations better than frequency-based methods

Average Word2Vec (AvgWord2Vec)

The Challenge: From Words to Documents

Word2Vec provides vector representations for individual **words**, but in practical NLP applications, we often need to represent entire **sentences** or **documents** as vectors. Average Word2Vec bridges this gap with a simple aggregation approach.

Concept

Average Word2Vec computes the mean (average) of all word vectors in a sentence or document to create a single fixed-length vector representation.

Mathematical Formula

For a document D containing words $w_1, w_2, ..., w_n$:

$\vec{D}_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \vec{w}_i$

Where:

- $\vec{D}_{\text{avg}}$ = averaged vector representation of the document/sentence
- n = total number of words in the document/sentence
- $\vec{w}_i$ = Word2Vec embedding vector for the i-th word
- $\sum_{i=1}^{n} \vec{w}_i$ = sum of all word vectors in the document

Interpretation: Each dimension of the document vector is the average of that dimension across all word vectors, creating a centroid representation in the embedding space.

Example Calculation

Consider the sentence: **"The cat sat on mat"**

Step 1: Obtain Word2Vec embeddings (assuming 3D vectors for illustration):

- $\vec{the} = [0.2, 0.5, 0.1]$
- $\vec{cat} = [0.8, 0.3, 0.6]$
- $\vec{sat} = [0.4, 0.7, 0.2]$
- $\vec{on} = [0.1, 0.4, 0.3]$
- $\vec{mat} = [0.7, 0.2, 0.5]$

Step 2: Calculate the average:
$\vec{sentence} = \frac{1}{5}([0.2, 0.5, 0.1] + [0.8, 0.3, 0.6] + [0.4, 0.7, 0.2] + [0.1, 0.4, 0.3] + [0.7, 0.2, 0.5])$

$\vec{sentence} = \frac{1}{5}[2.2, 2.1, 1.7] = [0.44, 0.42, 0.34]$

Weighted Average Word2Vec

To address the limitation of equal weighting, we can use **TF-IDF weights** to give more importance to significant words:

$\vec{D}_{\text{weighted}} = \frac{\sum_{i=1}^{n} \text{TF-IDF}(w_i) \cdot \vec{w}_i}{\sum_{i=1}^{n} \text{TF-IDF}(w_i)}$

Where:

- $\text{TF-IDF}(w_i)$ = TF-IDF score for word i (weight indicating word importance)
- $\vec{w}_i$ = Word2Vec embedding vector for word i
- The denominator normalizes by the sum of weights

Interpretation: Important words (higher TF-IDF scores) contribute more to the final document vector, while common words contribute less.

Advantages of AvgWord2Vec

1. **Fixed-length representation**: Produces consistent vector size regardless of document length
2. **Simple implementation**: Straightforward averaging operation
3. **Semantic awareness**: Leverages Word2Vec's semantic embeddings
4. **Pre-trained compatibility**: Works with existing pre-trained Word2Vec models
5. **Computational efficiency**: Fast computation compared to complex models

Limitations of AvgWord2Vec

1. **Word order loss**: "Dog bites man" and "Man bites dog" produce identical vectors
2. **Equal weighting**: All words contribute equally in basic version (unless using weighted variant)
3. **No syntactic structure**: Ignores grammar and sentence structure
4. **Context insensitivity**: Cannot capture context-dependent word meanings

Applications of AvgWord2Vec

- **Text Classification**: Sentiment analysis, spam detection, topic categorization
- **Document Similarity**: Computing semantic similarity between documents
- **Clustering**: Grouping similar documents together
- **Information Retrieval**: Document ranking and search
- **Baseline Models**: Quick prototyping before implementing complex architectures

Comprehensive Comparison of Word Embedding Methods

Aspect	Frequency-Based (BoW, TF-IDF)	Word2Vec (Individual Words)	AvgWord2Vec (Documents)	Weighted AvgWord2Vec
Dimensionality	High (sparse)	Low (dense)	Low (dense)	Low (dense)
Semantic capture	Poor	Excellent	Good	Good

Aspect	Frequency-Based (BoW, TF-IDF)	Word2Vec (Individual Words)	AvgWord2Vec (Documents)	Weighted AvgWord2Vec
Word order	Not preserved	N/A (single words)	Not preserved	Not preserved
Document representation	Native	Requires aggregation	Native	Native
Training time	Fast	Slower	Fast (uses pre-trained)	Fast (uses pre-trained)
Memory efficiency	Poor	Good	Good	Good
Context awareness	None	Strong	Moderate	Moderate
Word importance weighting	Manual (TF-IDF)	Implicit	Equal	TF-IDF weighted
Accuracy	Lower	Higher	Medium-High	Medium-High
Use case	Simple baselines	Word-level tasks	Document classification	Document classification

Prerequisites for Understanding Word2Vec

To fully understand and implement Word2Vec models, knowledge of the following concepts is essential:

- **Artificial Neural Networks (ANN)**: Architecture and forward/backward propagation
- **Loss Functions**: Cross-entropy, softmax
- **Optimizers**: Stochastic Gradient Descent (SGD), Adam
- **Backpropagation**: How gradients flow through the network

Conclusion

Word embeddings transform the way we represent text in machine learning models. While frequency-based methods like One-Hot Encoding, Bag of Words, and TF-IDF provide basic representations, they suffer from high dimensionality and inability to capture semantic relationships. Deep learning-based methods like Word2Vec (CBOW and Skip-gram) overcome these limitations by learning dense vector representations that encode semantic meaning, enabling more sophisticated natural language understanding tasks.

Note: Word2Vec and similar embedding techniques form the foundation for modern NLP applications including sentiment analysis, machine translation, question answering, and many other language understanding tasks.