Word Embeddings in NLP

1. What is Word Embedding?

Word embedding is a technique in Natural Language Processing (NLP) where words are represented as dense vectors in a continuous vector space. Unlike traditional sparse representations such as one-hot encoding, word embeddings capture semantic relationships between words by positioning similar words close to each other in the vector space.

Core Principles of Word Embeddings

Word embeddings are based on the distributional hypothesis, which states that words appearing in similar contexts tend to have similar meanings. This allows embeddings to capture:

- Semantic similarity: Words with similar meanings are represented by similar vectors
- Analogical relationships: Vector arithmetic can reveal relationships (e.g., king - man + woman ≈ queen)
- Contextual patterns: Words that appear in similar contexts are placed nearby in the vector space

Advantages Over Traditional Representations

- Dimensionality reduction: Instead of sparse vectors with vocabulary-sized dimensions, embeddings typically use 100-300 dimensions
- Generalization capability: By capturing semantic relationships, models can better generalize to unseen examples
- Rich semantic information: Embeddings encode multiple aspects of meaning simultaneously
- Continuous representation: The continuous nature of the vector space allows for measuring degrees of similarity

Applications in NLP

Word embeddings have become foundational components in various NLP tasks including:

- Sentiment analysis
- Text classification
- Named entity recognition
- Machine translation
- Question answering
- Document clustering

2. Explanation of Embedding Models

Word2Vec

Word2Vec, introduced by Mikolov et al. at Google in 2013, is one of the pioneering models for creating word embeddings. It comes in two main architectures:

- CBOW (Continuous Bag of Words)

- Approach: Predicts a target word from surrounding context words
- Training: Uses a neural network with a single hidden layer
- Efficiency: Generally trains faster than Skip-gram
- Performance: Works well for frequent words but may struggle with rare words

- Skip-gram

- Approach: Predicts surrounding context words given a target word
- Training: Also uses a neural network with a single hidden layer
- Efficiency: Slower to train but better for infrequent words
- Performance: Generally provides better quality embeddings, especially for rare words

Word2Vec uses negative sampling or hierarchical softmax techniques to make training computationally feasible on large corpora.

GloVe (Global Vectors for Word Representation)

GloVe, developed by Stanford researchers in 2014, combines the benefits of two main approaches in word embeddings:

- Count-based methods: Like Latent Semantic Analysis, which use global corpus statistics
- Predictive methods: Like Word2Vec, which learn vectors by predicting contexts

Key Characteristics

- Training objective: Based on global word-word co-occurrence statistics from the corpus
- Efficiency: Better utilizes statistical information than pure prediction-based methods
- Performance: Often captures both semantic and syntactic regularities well
- Pre-training: Commonly available as pre-trained vectors on various corpus sizes (e.g., 6B tokens used in your implementation)

FastText

FastText, developed by Facebook Research in 2016, extends Word2Vec by treating each word as a bag of character n-grams:

Key Innovations

- Subword information: Represents each word as a sum of its character n-grams
- OOV handling: Can generate embeddings for words not seen during training
- Morphological awareness: Better for languages with rich morphology (e.g., Turkish,

Finnish)
- Performance: Generally performs better for rare words and morphologically rich languages

Advantages Over Word2Vec and GloVe

- Robustness to typos: Similar spellings result in similar embeddings
- Better rare word handling: Can infer embeddings for infrequent or unseen words
- Smaller vocabulary size needed: Captures patterns at subword level rather than needing every word form

Comparison of Embedding Models

Feature	Word2Vec	GloVe	FastText
Training Approach	Predictive (context-based)	Count-based + Predictive	Predictive with subword units
Handling Rare Words	Limited	Limited	Strong
OOV Words	Cannot handle	Cannot handle	Can generate embeddings
Training Speed	Fast (CBOW), Slower (Skip-gram)	Moderate	Slower than Word2Vec
Semantic Capture	Good	Very good	Good
Morphological Awareness	Poor	Poor	Strong
Language Suitability	General	General	Especially good for morphologically rich languages

3. Problem Definition and Solutions

Problem Definition

In this study, we addressed the binary sentiment analysis problem using the IMDb movie reviews dataset. The goal was to classify movie reviews as either positive or negative based on their text content. This is a classic natural language processing (NLP) task that requires understanding the sentiment expressed in human-written text.

Dataset Description

We used the IMDb reviews dataset from TensorFlow Datasets (TFDS), which contains:

- 25,000 training examples
- 25,000 test examples
- Binary labels (0 for negative, 1 for positive)
- Reviews of varying lengths and complexity

Solution Approaches

We implemented two distinct approaches to solve this sentiment classification problem:

Approach 1: TF-IDF Vectorization (Baseline)

Our baseline approach utilized the traditional TF-IDF (Term Frequency-Inverse Document Frequency) method for text representation, followed by a Logistic Regression classifier.

Implementation Details:

1. **Data Preprocessing**:
    - Converted all text to lowercase
    - Removed HTML tags using regular expressions
    - Eliminated non-alphabetic characters
    - Removed extra whitespace
    - Tokenized text using NLTK's word_tokenize
    - Removed English stopwords
    - Removed short words (length ≤ 1)
2. **Feature Extraction**:
    - Used scikit-learn's TfidfVectorizer
    - Limited to the top 10,000 features (max_features=10000)
    - Included both unigrams and bigrams (ngram_range=(1, 2))
    - Created sparse matrices of shape (25000, 10000) for both training and test sets
3. **Model Training**:
    - Trained a Logistic Regression model (sklearn.linear_model.LogisticRegression)
    - Used liblinear solver with max_iter=1000 and C=1.0 regularization
    - Fixed random_state=42 for reproducibility
4. **Evaluation**:
    - Calculated accuracy, precision, recall, and F1-score
    - Generated confusion matrix
    - Achieved 88.66% accuracy on the test set

Approach 2: Word Embeddings with GloVe

Our second approach leveraged pre-trained GloVe (Global Vectors for Word Representation) embeddings to create document-level representations.

Implementation Details:

1. **Data Preprocessing**:
    - Used the same preprocessing pipeline as the TF-IDF approach
2. **Embedding Loading**:

- Downloaded and extracted GloVe pre-trained vectors (glove.6B.100d.txt)
- Loaded 400,000 word vectors with 100 dimensions each into a Python dictionary
- Used embedding_dim=100 for all vectors

3. **Document Vector Creation**:
   - Created a document vector for each review by averaging the embeddings of all words
   - Implemented get_doc_vector function to:
     - Split preprocessed text into tokens
     - Retrieve embedding vectors for each word from the GloVe dictionary
     - Calculate the mean of all found word vectors
     - Return a zero vector for documents with no words in the embedding dictionary
   - Generated embedding matrices of shape (25000, 100) for both training and test sets
4. **Model Training**:
   - Trained a Logistic Regression model with the same parameters as the baseline
   - Used the dense document vectors as input features
5. **Evaluation**:
   - Calculated the same metrics as the baseline approach
   - Achieved 79.98% accuracy on the test set

Performance Visualization and Analysis

We created visualizations to compare the performance of both approaches:

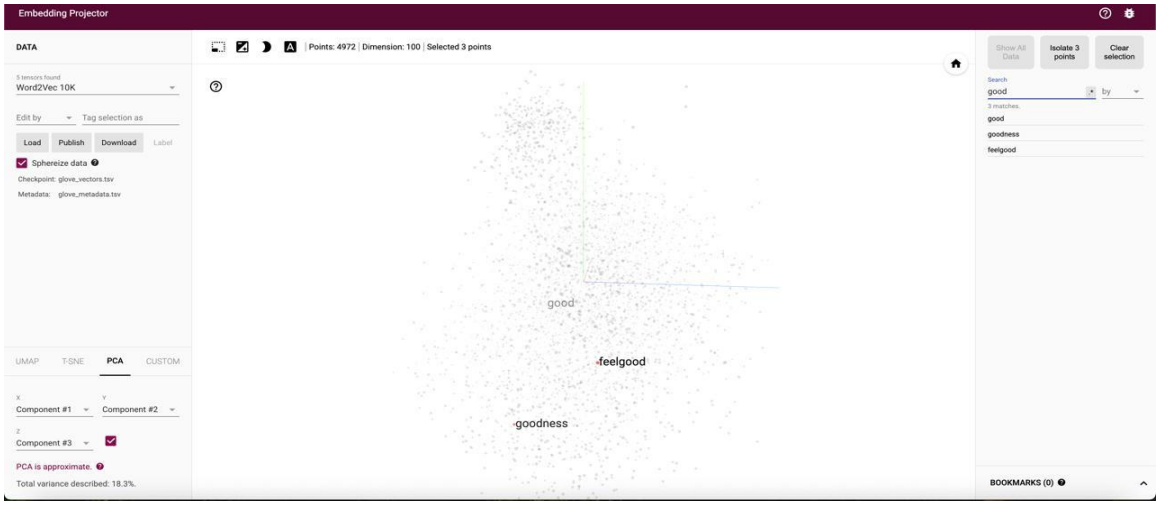
1. **Confusion Matrices**: Generated heatmap visualizations for both models
   - TF-IDF Model: Used a blue color palette
   - GloVe Model: Used a green color palette
2. **Performance Metrics Comparison**:
   - Created a DataFrame comparing key metrics (Accuracy, F1-Score) for both models
   - Generated bar charts for visual comparison
3. **Analysis**:
   - The TF-IDF model outperformed the GloVe embedding model by 8.68 percentage points
   - Analyzed potential reasons for this performance difference, including:
     - Information loss in simple averaging of embeddings
     - OOV (Out-of-Vocabulary) word handling
     - The advantage of n-grams in TF-IDF for capturing important word combinations
     - Model compatibility with different feature types

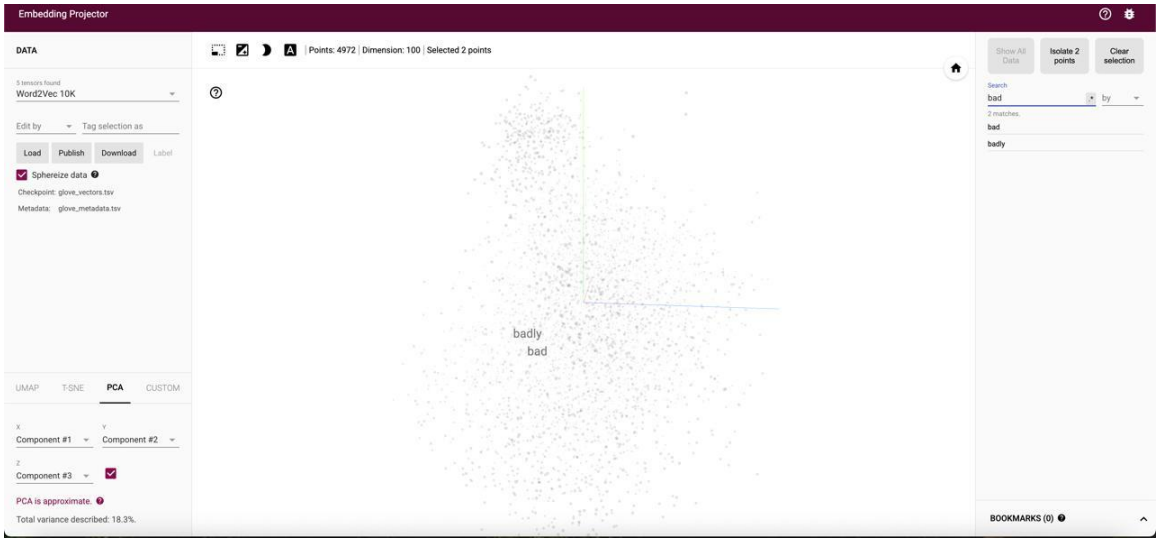
Computational Performance

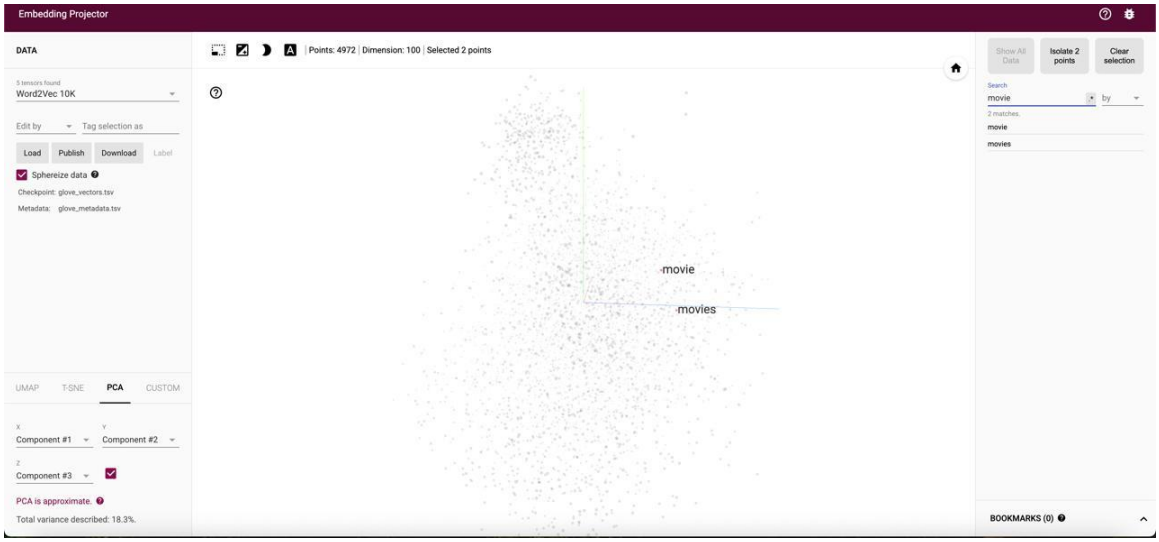
We also measured the computational efficiency of both approaches:

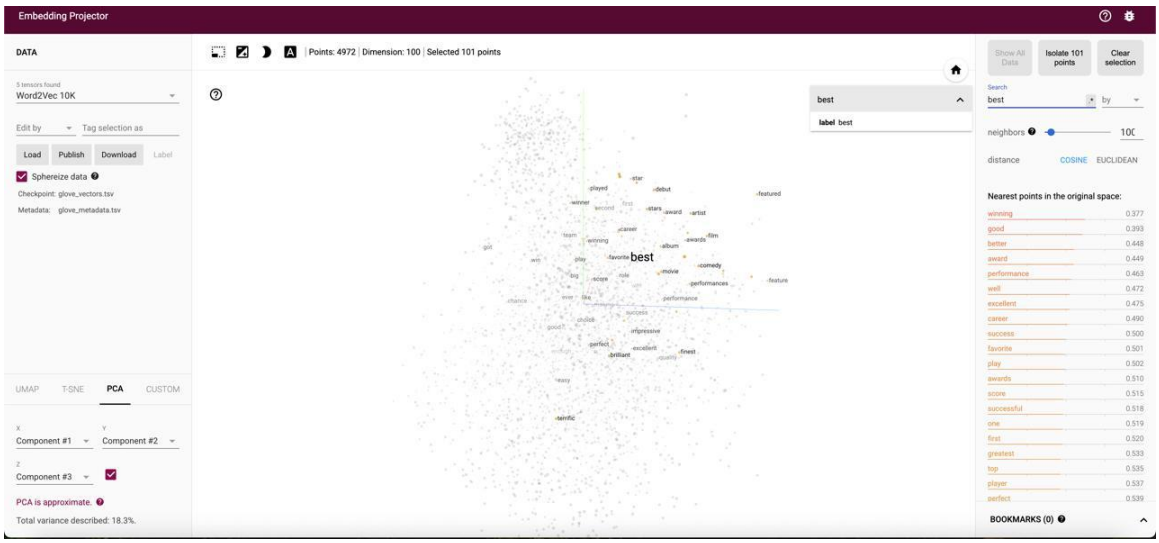
1. **TF-IDF Processing**:
    - Text preprocessing: 46.37 seconds
    - TF-IDF vectorization: 26.83 seconds
    - Model training: 0.88 seconds
    - Prediction: 0.01 seconds
2. **GloVe Embedding Processing**:
    - GloVe loading: 11.61 seconds
    - Document vector creation: 7.80 seconds
    - Model training: 1.06 seconds
    - Prediction: 0.01 seconds

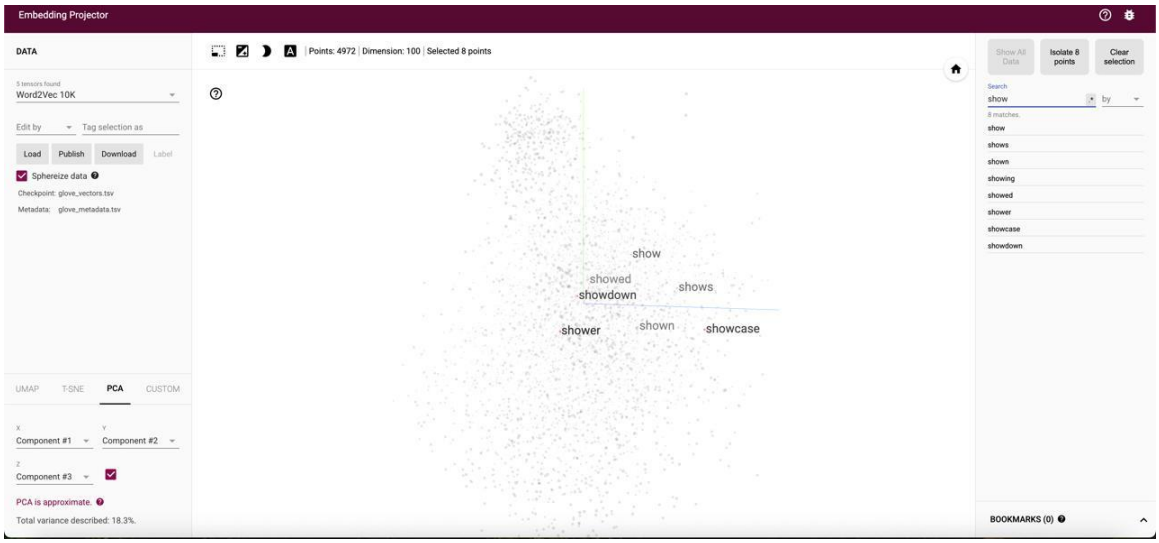
3. Visualization Using Tensorflow






Embedding Projector
DATA
Word2Vec 10K
Load
Publish
Download
Sphereize data
Checkpoint: glove_vectors.tsv
Metadata: glove_metadata.tsv
UMAP
T-SNE
PCA
CUSTOM
Component #1
Component #2
Component #3
PCA is approximate.
Total variance described: 18.3%.
Points: 4972 | Dimension: 100 | Selected 2 points
movie
movies
Show All Data
Isolate 2 points
Clear selection
Search
movie
2 matches.
movie
movies
BOOKMARKS (0)


Embedding Projector
DATA
Word2Vec 10K
Load
Publish
Download
Sphereize data
Checkpoint: glove_vectors.tsv
Metadata: glove_metadata.tsv
UMAP
T-SNE
PCA
CUSTOM
Component #1
Component #2
Component #3
PCA is approximate.
Total variance described: 18.3%.
Points: 4972 | Dimension: 100 | Selected 101 points
best
label best
Show All Data
Isolate 101 points
Clear selection
Search
best
neighbors
distance
COSINE
EUCLIDEAN
Nearest points in the original space:
winning 0.377
good 0.393
better 0.448
award 0.449
performance 0.463
well 0.472
excellent 0.475
career 0.490
success 0.500
favorite 0.501
play 0.502
awards 0.510
score 0.515
successful 0.518
one 0.519
first 0.520
greatest 0.533
top 0.535
player 0.537
perfect 0.539
BOOKMARKS (0)


Embedding Projector
DATA
Word2Vec 10K
Load
Publish
Download
Sphereize data
Checkpoint: glove_vectors.tsv
Metadata: glove_metadata.tsv
UMAP
T-SNE
PCA
CUSTOM
Component #1
Component #2
Component #3
PCA is approximate.
Total variance described: 18.3%.
Points: 4972 | Dimension: 100 | Selected 8 points
show
showed
showdown
shows
shower
shown
showcase
Show All Data
Isolate 8 points
Clear selection
Search
show
8 matches.
show
shows
shown
showing
showed
shower
showcase
showdown
BOOKMARKS (0)

5. Group Member Contributions

Ezgi Altıok (34%): Responsible for data preprocessing, TF-IDF implementation, and performance evaluation

Göktürk Can (33%): Implemented GloVe embeddings, visualization, and results analysis

Mehmet Sefa Toksoy (33%): Wrote theoretical sections, created presentation materials, and managed GitHub repository

7. References

1. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
2. Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In EMNLP (pp. 1532-1543).
3. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. TACL, 5, 135-146.
4. TensorFlow Datasets. IMDb Reviews. Retrieved from https://www.tensorflow.org/datasets/catalog/imdb_reviews
5. Stanford NLP Group. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/
6. Maas, A. L., et al. (2011). Learning word vectors for sentiment analysis. In ACL (pp. 142-150).
7. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
8. TensorFlow. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/
9. Natural Language Toolkit (NLTK). https://www.nltk.org/