Word Vector Embeddings via Co-occurrence Matrix and SVD

A Study in Count-Based NLP Models

Submitted by: Mohd Umair
Roll No.: 22336
GitHub Link : Assignment

Course: NLP with Deep Learning
Submitted to: Prof. Neha Sharma

August 29, 2025

Abstract

This report outlines the process of generating dense word embeddings from a text corpus using a count-based vector space model. The methodology involves preprocessing a custom corpus, constructing a word co-occurrence matrix, and then applying dimensionality reduction using Truncated Singular Value Decomposition (SVD) to produce low-dimensional word vectors. The resulting 2D embeddings are visualized to demonstrate how the model captures semantic relationships between words. This work provides a practical implementation of fundamental NLP concepts for word representation.

1 Introduction

In Natural Language Processing (NLP), word embeddings are a crucial technique for representing words as numerical vectors. These dense vectors capture the semantic meaning, context, and relationships between words, which is a significant improvement over sparse representations like one-hot encoding.

This project implements a classic count-based method for generating word embeddings. The core idea is that words that appear in similar contexts are semantically related. We quantify this by building a co-occurrence matrix, which tracks how frequently words appear near each other. This high-dimensional, sparse matrix is then reduced to a low-dimensional, dense representation using SVD.

2 Methodology and Implementation

The implementation is structured as a pipeline of functions, each performing a distinct step from text preprocessing to visualization.

2.1 Corpus Definition

The model was trained on a custom corpus focused on financial news to analyze the resulting semantic relationships.

```
corpus = """
Global markets rallied on Thursday after the U.S. Federal Reserve
    signaled that it might ease interest rates.
Investors welcomed the news as stocks rose across Europe and Asia.
The central banks statement also pointed to continued uncertainty in
     the global economy.
"""
```

Listing 1: Custom corpus for the model.

2.2 Text Preprocessing and Tokenization

The initial step involves cleaning the corpus by converting it to lowercase and extracting a sequence of word tokens. Punctuation and special characters are removed.

```
import re

def preprocess_text(corpus):
    # Lowercase and extract words
    tokens = re.findall(r'\b\w+\b', corpus.lower())
    return tokens
```

Listing 2: Function for tokenizing the corpus.

2.3 Vocabulary Extraction

From the list of tokens, a unique, sorted vocabulary is created. This defines the dimensions of the co-occurrence matrix.

```
def get_distinct_words(tokens):
    distinct_words = sorted(set(tokens))
    return tokens, distinct_words
```

Listing 3: Function to create the vocabulary.

2.4 Co-occurrence Matrix Construction

A co-occurrence matrix M is built to quantify how often words appear together. For a vocabulary of size V, M is a $V \times V$ matrix where M_{ij} stores the number of times word j appears in the context window of word i. A window size of $n = 2$ was used.

```
import numpy as np

def build_cooccurrence_matrix(tokens, vocab, window_size=2):
    vocab_to_index = {word: i for i, word in enumerate(vocab)}
    cooccurrence_matrix = np.zeros((len(vocab), len(vocab)))

    for idx, word in enumerate(tokens):
        word_idx = vocab_to_index[word]
        start = max(0, idx - window_size)
        end = min(len(tokens), idx + window_size + 1)
        for i in range(start, end):
            if i != idx:
                context_word = tokens[i]
                context_idx = vocab_to_index[context_word]
                cooccurrence_matrix[word_idx][context_idx] += 1

    return cooccurrence_matrix, vocab_to_index
```

Listing 4: Function to build the co-occurrence matrix.

2.5 Dimensionality Reduction with Truncated SVD

The co-occurrence matrix is typically large and sparse. To create dense embeddings, we reduce its dimensions. Truncated Singular Value Decomposition (SVD) is used for this task, as it is well-suited for large sparse matrices. It factorizes the matrix and allows us to select the top k components that capture the most variance in the data. We reduce the matrix to $k = 2$ dimensions for plotting.

```
from sklearn.decomposition import TruncatedSVD

def reduce_dimensions(matrix, k=2):
    svd = TruncatedSVD(n_components=k)
    reduced = svd.fit_transform(matrix)
    return reduced
```

Listing 5: Function to reduce dimensions using SVD.

2.6 Visualization of Embeddings

Finally, the 2D word embeddings are plotted on a scatter graph to allow for visual inspection of the captured semantic relationships.

```
import matplotlib.pyplot as plt

def plot_embeddings(embeddings, vocab, num_words=50):
    plt.figure(figsize=(14, 10))
    for i, word in enumerate(vocab[:num_words]):
        x, y = embeddings[i]
        plt.scatter(x, y)
        plt.text(x + 0.01, y + 0.01, word, fontsize=9)
    plt.title("2D Word Embeddings (Top {} Words)".format(num_words))
    plt.grid(True)
    plt.show()
```

Listing 6: Function for plotting the 2D vectors.

3 Results and Discussion

The model processes the financial corpus and produces a 2D vector for each word in its vocabulary. The resulting plot (Figure 1) allows for a visual analysis of these vectors. It is expected that words with related financial or economic meanings, such as ‘markets‘, ‘stocks‘, ‘investors‘, and ‘economy‘, will appear in close proximity to one another in the vector space. This clustering would validate the model’s ability to learn meaningful semantic relationships from the co-occurrence statistics in the text.

4 Conclusion

This project successfully implemented a pipeline for generating and visualizing word embeddings using a co-occurrence matrix and Truncated SVD. The methodology demonstrates a fundamental, count-based approach to word vector representation in NLP. The results show that even with a small corpus, it is possible to capture and visualize meaningful semantic relationships between words, providing a solid foundation for understanding more complex embedding models like Word2Vec and GloVe.

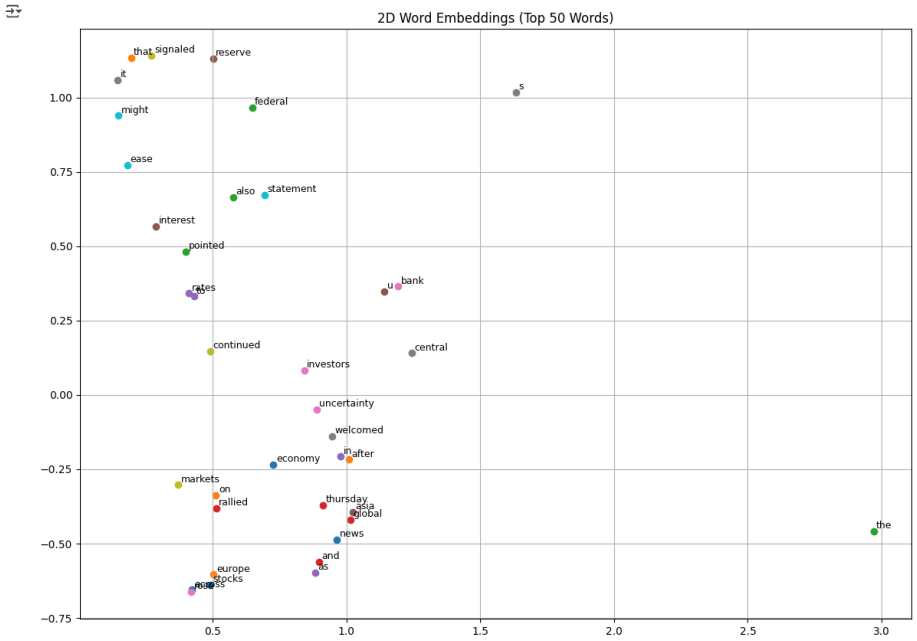

Figure 1: A 2D visualization of the word embeddings. Words with similar meanings or contexts are expected to appear closer together.

A Full Python Implementation

```
# -*- coding: utf-8 -*-
"""NLPwithDL.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1vwq4R1I0aI8x1xxy_t822xNALg85lF7M

Define Custom Corpus
"""
import re
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD

corpus = """
Global markets rallied on Thursday after the U.S. Federal Reserve signaled that it might ease interest rates.
Investors welcomed the news as stocks rose across Europe and Asia.
The central banks statement also pointed to continued uncertainty in the global economy.
"""

"""Tokenize and Clean Text"""

def preprocess_text(corpus):
    # Lowercase and extract words
    tokens = re.findall(r'\b\w+\b', corpus.lower())
    return tokens
```

```python

tokens = preprocess_text(corpus)
print("Tokens:", tokens)

"""Get Distinct Words (Vocabulary)"""

def get_distinct_words(tokens):
    distinct_words = sorted(set(tokens))
    return tokens, distinct_words

tokens, vocab = get_distinct_words(tokens)
print("Vocabulary size:", len(vocab))

"""Build Co-occurrence Matrix"""

def build_cooccurrence_matrix(tokens, vocab, window_size=2):
    vocab_to_index = {word: i for i, word in enumerate(vocab)}
    cooccurrence_matrix = np.zeros((len(vocab), len(vocab)))

    for idx, word in enumerate(tokens):
        word_idx = vocab_to_index[word]
        start = max(0, idx - window_size)
        end = min(len(tokens), idx + window_size + 1)
        for i in range(start, end):
            if i != idx:
                context_word = tokens[i]
                context_idx = vocab_to_index[context_word]
                cooccurrence_matrix[word_idx][context_idx] += 1

    return cooccurrence_matrix, vocab_to_index

co_matrix, word2idx = build_cooccurrence_matrix(tokens, vocab,
    window_size=2)

"""Reduce Dimensions (SVD)"""

def reduce_dimensions(matrix, k=2):
    svd = TruncatedSVD(n_components=k)
    reduced = svd.fit_transform(matrix)
    return reduced

reduced_embeddings = reduce_dimensions(co_matrix, k=2)

"""Plot 2D Embeddings"""

def plot_embeddings(embeddings, vocab, num_words=50):
    plt.figure(figsize=(14, 10))
    for i, word in enumerate(vocab[:num_words]):
        x, y = embeddings[i]
        plt.scatter(x, y)
        plt.text(x + 0.01, y + 0.01, word, fontsize=9)
    plt.title("2D Word Embeddings (Top {} Words)".format(num_words))
    plt.grid(True)
    plt.show()

plot_embeddings(reduced_embeddings, vocab, num_words=50)
```

Listing 7: The complete Python script.