Vector Semantics and Word Embeddings

Ritwika Das Gupta (2348049)
Soham Chatterjee (2348062)
Sayantan Ray (2348057)

Submitted to Prof Dr Saleema J S

Department of Statistics and Data Science
CHRIST (Deemed to be University)
September 03, 2024

Abstract

This is a summary discussing vector semantics and word embeddings, techniques in NLP that bridge the gap between human languages and machines. Vector semantics map words and phrases onto numeric vectors in a continuous space so as to capture the meaning and relations among them. It proceeds to discuss some traditional methods, like sparse vector embeddings that comprise one-hot encoding and the BoW model; it points at the limitation regarding dimensionality and interpretability. More sophisticated TF-IDF and PPMI methods were proposed to improve the representational power of these models. Other work described in the paper is the dense vector embeddings that encode words as high-dimensional vectors by learning semantically and contextually similar words to have similar vector representations. These embeddings changed NLP due to the meaningful word representations they introduced, which in turn helped tasks such as text classification, sentiment analysis, and machine translation. This summary study is done in to show each of these models as important steps in word embeddings for progress in NLP.

Find the implementation of our understanding in python: here

Department of Statistics & Data Science, Christ University

Contents

List of Figures

1 Introduction

"Nets are for fish. Once you get the fish, you can forget the net. Words are for meaning; Once you get the meaning, you can forget the words."

Living in the digital world with a whole glut of text data, the goal of processing and understanding human language is one of the most challenging jobs. There is one issue that makes things even more complicated: each word has more than one meaning, depending on the context. In reality, NLP requires vector semantics and word embeddings to fill the language gap between human expression and machine understanding.

1.1 What are vector semantics in NLP?

Suppose we don't know what the Hindi word **"samosa"** means, but you encounter it in the following sentences or contexts: (6.1) "The samosa is crispy and spicy." (6.2) "I had a delicious samosa for a snack." (6.3) "Samosas are often served with chutney." Additionally, we have seen many of these context words occurring in similar contexts, such as: (6.4) "The crispy and spicy empanada was a hit." (6.5) "I enjoyed a savory pastry filled with spiced potatoes." (6.6) "The spring rolls were served with a tangy dipping sauce." The fact that we can infer that "samosa" is a type of savory, crispy, and spiced food item, often enjoyed as a snack or appetizer and served with condiments.
The fact that words like "crispy," "spicy," "savory pastry," and "served with chutney" also occur around other food items like "empanada" and "spring rolls" can help us discover the similarity between "samosa" and these other food items. **This is the crux of vector semantics.**
This was the theory laid down by the skeptic philosopher Ludwig Wittgenstein. (1) To realize Wittgenstein's intuition, linguists Joos, Harris, and Firth (the linguistic distributionalists) developed a particular concept: define a word by its environment or distribution in language use.

What is semantic? According to the Oxford Dictionary, "semantics" means – of or relating to meaning in language. What is a vector? We can think of it as a numerical representation of a list of things describing an item. Combining these two, Vector Semantics is a mathematical approach to representing the meanings of words and phrases as vectors in a continuous vector space. Traditionally, words were treated as discrete symbols with no inherent relationship to each other. This posed significant challenges for tasks like text classification, sentiment analysis, and machine translation, where understanding the subtle relationships between words is crucial.

1.2 Summary/ Basic idea:

What we understood so far is vector semantics dealing with measuring the semantic similarity of words in terms of the similarity of the contexts in which they appear. How? Represent words as vectors such that — each vector element (dimension) corresponds to a different context — the vector for any particular word captures how strongly it is associated with each context. Compute the semantic similarity of words as the similarity of their vectors.**Vectors for representing words are generally called embeddings**, because the word is embedded in a particular vector space.

2 Word Embeddings

Word Embedding is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meanings to have a similar representation.(2)

2.1 Need for Word Embedding?

To reduce dimensionality To use a word to predict the words around it. Inter-word semantics must be captured. How are Word Embeddings used? They are used as input to machine learning models. Take the words —-> Give their numeric representation —-> Use in training or inference. To represent or visualize any underlying patterns of usage in the corpus that was used to train them.(3)

3 Sparse Vector Embedding

One of the very first attempts made to encode meanings of words in vector space was that of sparse vectors. These representations are highly dimensional, with **most of their dimensions assuming the value zero**, and hence very sparse.

3.0.1 Simplest and earliest method- One hot encoding

In this encoding scheme, each word in the vocabulary is represented as a unique vector, where the dimensionality of the vector is equal to the size of the vocabulary. The vector has all elements set to 0, except for the element corresponding to the index of the word in the vocabulary, which is set to 1. Fig 3.1 shows a simple example that we did.

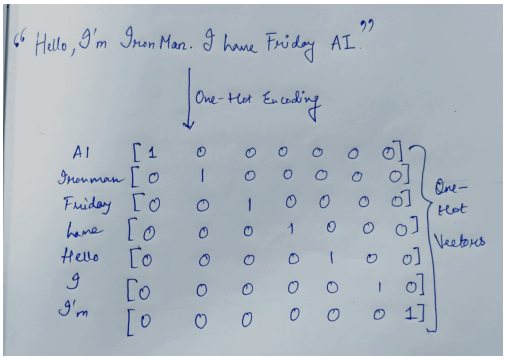

Figure 3.1: One Hot vectors : example

3.0.2 Disadvantage of one hot encoding

1. Dimensionality: It can lead to a high-dimensional feature space, especially when dealing with large categorical variables

2. Sparse Representation: It produces sparse matrices with mostly zeros, which can lead to memory inefficiency(4) and computational overhead.

3.1 Bag of Words

Bag of Words (BoW) featurization is often perceived as a beginner-level text processing technique due to its apparent simplicity in counting words across a text corpus. However, the underlying concepts are more intricate, particularly when viewed through the lens of vector spaces. (5)

3.1.1 Understanding Vector Spaces in Bag of Words

A vector space is a multi-dimensional space where each axis represents a feature, and in the context of a Bag of Words model, each unique word in the text corpus forms a separate dimension. If a text corpus contains n unique words, the corresponding vector space has n dimensions. Each document in the corpus is then plotted as a point in this space, with the position along each dimension determined by the frequency of the word corresponding to that dimension.

For example, consider a corpus with the following two documents:

- **Document 1**: NLP is fun, yeah! let's go.
- **Document 2**: yeah, let's go Soham, yeah!

For simplicity, let's focus on a subset of the vocabulary: *yeah*, *fun*, and *go*.

3.1.2 Vector Representation of Documents

In **Document 1**, the words *yeah*, *fun*, and *go* appear once each, so the document can be represented as the vector:

$$\mathbf{d}_1 = (1, 1, 1)$$

In **Document 2**, the word *yeah* appears twice, while *go* and *fun* each appear 1 and 0 times respectively. Therefore, the vector representation for this document is:

$$\mathbf{d}_2 = (2, 1, 1)$$

These document vectors are points in the three-dimensional vector space, with coordinates corresponding to the frequency of each word.

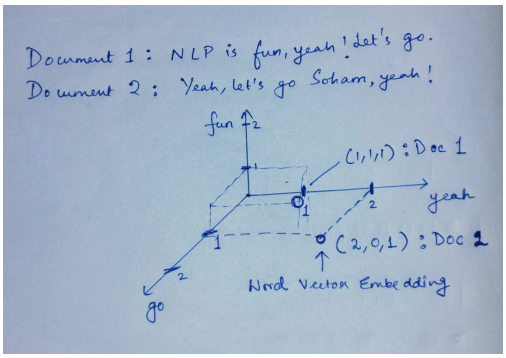

Figure 3.2: 3D Vector Space Representation of Documents 1 and 2

3.2 Vectors and Documents

Vector or distributional models of meaning are generally based on a co-occurrence matrix, which represents how often words co-occur. In this section, we will explore two popular matrices: the term-document matrix and the term-term matrix.

3.2.1 Term-Document Matrix

In a *term-document matrix*, each row represents a word in the vocabulary, and each column represents a document from a collection. The value in each cell of the matrix is the frequency of the corresponding word in the corresponding document.(1) An example of a term-document matrix for four words in four movie descriptions (two comedies and two thrillers) is shown in Table 1.

	Fall Guy	**Golmaal**	**Inception**	**Parasite**
good	36	41	34	24
intense	1	2	21	37
hilarious	16	37	1	4
suspense	2	1	29	49

Table 1: Term-Document Matrix for four words in four movie descriptions.

In this matrix, each movie description is represented as a vector of word counts. For example, **Fall Guy** is represented by the vector [36, 1, 16, 2], where the dimensions correspond to the words *good*, *intense*, *hilarious*, and *suspense*. The dimensionality of the vector space corresponds to the size of the vocabulary, which in this example is 4.

3.3 TF-IDF: Weighing Terms in the Vector

The co-occurrence matrix above 1 represents each cell by the frequency of a word in a particular document (movie description). However, raw frequency is not always the best measure of a word's importance in a document. Raw frequency tends to be skewed and not very discriminative,

especially when dealing with commonly occurring words. For example, frequent words like *the* or *it* are not very informative in distinguishing between different documents.

3.3.1 Term Frequency (TF)

Mathematically, it can be represented as:

$$\text{TF}(t, d) = \text{count}(t, d)$$

However, we often adjust the raw frequency using a logarithmic scale to reduce the impact of high-frequency words:

$$\text{TF}(t, d) = \begin{cases} 1 + \log_{10}(\text{count}(t, d)) & \text{if count}(t, d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

3.3.2 Inverse Document Frequency (IDF)

The second component of TF-IDF is the *inverse document frequency* (IDF), which gives higher weight to words that appear in fewer documents. The idea is to reduce the weight of common words that appear across many documents and emphasize words that are unique to specific documents.(6)

$$\text{IDF}(t) = \log_{10}\left(\frac{N}{df_t}\right)$$

3.3.3 Combining TF and IDF

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

This combined metric helps to balance the importance of word frequency within a document while also considering how common the word is across the entire document collection.

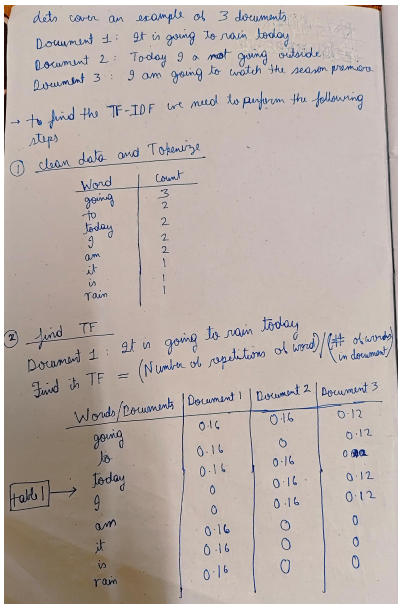

Figure 3.3: Finding Tf

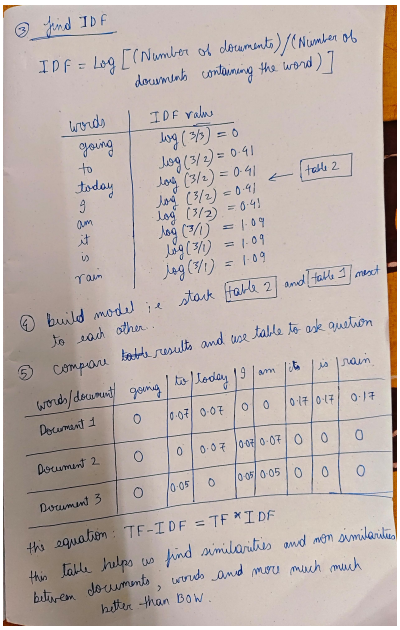

Figure 3.4: Finding IDF

Figure 3.5: Here is an example for calculating TF-IDF scores.

An alternative to the tf-idf weighting function is **Positive Pointwise Mutual Information (PPMI)**, which is used for term-term matrices when the vector dimensions represent words rather than documents. PPMI is based on the idea that the best way to evaluate the relationship between two words is to measure how frequently they appear together compared to what we would expect by chance. It quantifies the likelihood of two events x and y occurring together compared to their independent occurrences:

$$I(x;y) = \log_2 \frac{P(x;y)}{P(x)P(y)} \tag{1}$$

For a target word w and a context word c, the pointwise mutual information is defined as:

$$\text{PMI}(w;c) = \log_2 \frac{P(w;c)}{P(w)P(c)} \tag{2}$$

3.4 Measuring Similarity of words using Cosine

To measure similarity between two target words v and w, we need a metric that takes two vectors (of the same dimensionality, either both with words as dimensions, hence of length $|V|$, or both with documents as dimensions, of length $|D|$) and gives a measure of their similarity.

The cosine similarity metric between two vectors v and w thusan be computed as:

$$\text{cosine}(v, w) = \frac{v \cdot w}{\|v\|\|w\|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{3}$$

The cosine similarities are calculated as follows:

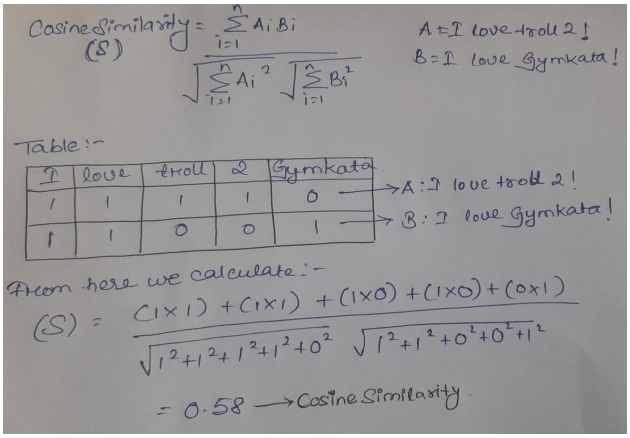

Figure 3.6: Cosine similarity

4 Dense Vector Embedding

Consider the sparse vectors that represent syntax, in which case dense vectors would be numerical representations of semantic meaning. We take words and encode them into very dense, high-dimensional vectors. The abstract meaning and relationship of words are numerically encoded.

4.1 Word2Vec Method

Word2Vec is a popular algorithm used to create word embeddings, which are vector representations of words that capture semantic similarities and contextual relationships. The method trains a neural network on a large corpus to associate words with numerical vectors based on their context.

The architecture of Word2Vec consists of:

- **Input Layer**: A one-hot encoded vector $\mathbf{x}$ representing the input word.
- **Hidden Layer**: A hidden layer with N neurons, where N is the dimensionality of the embedding space. The weight matrix $\mathbf{W}$ between the input and hidden layer stores the word embeddings.
- **Output Layer**: Outputs a probability distribution over the entire vocabulary using the softmax function.

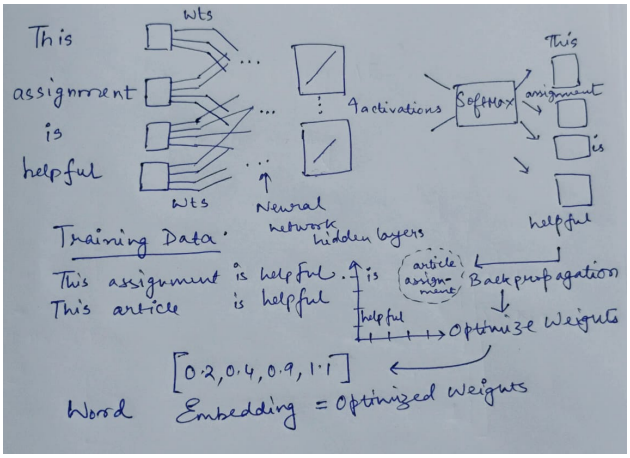

Figure 4.1: Word2Vec algorithm

In **CBOW**, the model predicts a target word w_t given the context words $w_{t-k}, \ldots, w_{t+1}, \ldots, w_{t+k}$, where k is the context window size. The objective is to maximize the probability:

$$P(w_t | w_{t-k}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k}) \tag{4}$$

In **Skip-Gram**, the model predicts the context words given a target word w_t. The objective is to maximize the same probability:

Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.

To reduce the computational complexity, Word2Vec uses **Negative Sampling.** Instead of updating the weights for all possible outputs, the model selects a few negative samples and updates only those.

4.2 GloVE- Global Vectors

GloVe is a word embedding model that captures both local and global statistical information about words within a corpus. It constructs a word-context matrix where each entry represents the co-occurrence frequency of a word pair within a specific context window. The core idea behind GloVe is to factorize this word-context co-occurrence matrix to obtain word vectors that encode semantic meanings.

The matrix factorization objective is defined as:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(\mathbf{w}_i^\top \mathbf{c}_j + b_i + b_j - \log X_{ij} \right)^2 \tag{5}$$

where X_{ij} is the co-occurrence count of words i and j, $\mathbf{w}_i$ and $\mathbf{c}_j$ are the word and context vectors, respectively, b_i and b_j are the biases, and $f(X_{ij})$ is a weighting function.

GloVe efficiently captures semantic relationships, such as word analogies, by modeling the global corpus statistics, making it effective for various NLP tasks.

4.3 FastText

FastText, developed by Facebook AI Research, extends the Word2Vec model by incorporating subword information. Unlike models that treat words as atomic units, FastText represents each word as a bag of character n-grams. For instance, the word *apple* might be represented by the n-grams *ap*, *pp*, *ple*, and so on. This approach allows FastText to generate meaningful word embeddings for rare and out-of-vocabulary words.

The model's objective function is similar to Word2Vec, but with the inclusion of subword information:

$$J = -\sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j}|w_t) \tag{6}$$

where $P(w_{t+j}|w_t)$ considers not just the word w_t but also its constituent n-grams.

FastText is particularly effective for languages with rich morphology and provides robust word representations that handle rare and unseen words better than traditional models.

5 Limitations and Biases

Even though word embeddings have greatly improved how we handle language tasks, they have some clear drawbacks. A major problem is the bias in models like Word2Vec and GloVe, which often reflect the stereotypes found in their training data. For example, **these models may link "doctor" with "man" and "nurse" with "woman,"** reinforcing unfair assumptions. Or like **'smart', 'wise', 'thoughtful', 'resourceful') had a higher cosine with male than female words**, and showed that this bias has been slowly decreasing since 1960.This is especially concerning when these models are used in real-life decisions, like hiring or lending. While there are ways to reduce these biases, they are not perfect, and some biases still remain hidden. More research and better data are needed to fully address these issues.

As far as limitations, **Word2Vec may not effectively handle polysemy, where a single word has multiple meanings.** The model might average or mix the representations of different senses of a polysemous word. Word2Vec also treats words as atomic units and does not capture subword information

6 Conclusion

Word embeddings, particularly advanced models like BERT, have proven superior to older methods like Bag-of-Words and Word2Vec. These newer models understand context much better,

making them highly effective in various applications. In the future, both sparse and dense word embeddings are expected to play crucial roles in real-world scenarios, from personal assistants to medical diagnostics, enhancing how machines understand human language. **Skip-gram works well** with handling vast amounts of text data and is found to **represent rare words well**. **GloVe embeddings are widely used** in NLP tasks, such as **text classification, sentiment analysis, machine translation** Word embeddings contribute to the success of **question answering systems** by enhancing the understanding of the context in which questions are posed and answers are found.and more. As these technologies evolve, they will continue to open new possibilities, making interactions with AI more natural and helpful.

7 Roles

1. **Soham Chatterjee** (2348062): Developed the **introduction**, explained the **vector embedding techniques using BOW, word2vec including skipgram**,examined **vector and document relationships**, and combined and compiled the final **LaTeX report**

2. **Ritwika Das Gupta** (2348049): Analyzed **sparse vector embedding** methods for vectors and documents, evaluated **cosine similarity**, identified **limitations**, formulated **conclusions,** and **implementation in python**

3. **Sayantan Ray** (2348057): Wrote about **vector semantics**, explored **TFIDF and BOW techniques**, **gloVE and fasttext, Pointwise mutual information** and implementation in python

References

[1] D. Jurafsky, "Speech and language processing," 2000.

[2] "Understanding nlp word embeddings — text vectorization | by prabhu raghav | towards data science." https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223. (Accessed on 09/03/2024).

[3] "Word embeddings in nlp: A complete guide." https://www.turing.com/kb/guide-on-word-embeddings-in-nlp. (Accessed on 09/03/2024).

[4] "Text vectorization and word embedding | guide to master nlp (part 5)." https://www.analyticsvidhya.com/blog/. (Accessed on 09/03/2024).

[5] "What is bag of words? | ibm." https://www.ibm.com/topics/bag-of-words#:~:text=Sparsity.,given%20vector%20may%20be%20zero. (Accessed on 09/03/2024).

[6] "Understanding tf-idf (term frequency-inverse document frequency) geeksforgeeks." https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/. (Accessed on 09/03/2024).