The Mathematics of Words

Miquel Noguer i Alonso
Artificial Intelligence Finance Institute

December 14, 2024

Abstract

The mathematical study of word embeddings reveals how meaning can be systematically represented in continuous vector spaces. Word embeddings revolutionized natural language processing by transforming discrete words into continuous vector spaces that capture semantic relationships. This paper provides a mathematical framework for understanding how different embedding techniques construct and utilize these meaning spaces. We analyze the geometric properties that allow embeddings to represent semantic relationships through vector operations, from simple word-level mappings to complex contextual representations. Starting with foundational models like Word2Vec and GloVe, we develop the theoretical principles behind meaning spaces and examine how they extend to contextual, multilingual, and knowledge-based embeddings. Through rigorous mathematical analysis, we demonstrate how different embedding approaches construct complementary aspects of meaning space and how these representations enable sophisticated natural language understanding tasks.

1 Introduction

Word embeddings are a cornerstone of modern natural language processing (NLP) techniques. They represent words as dense vectors in a continuous vector space, capturing semantic and syntactic similarities. This document delves into the advanced mathematical foundations of word embeddings, their training methods, and applications. By understanding the mathematical underpinnings, we can better appreciate the power and versatility of word embeddings in various NLP tasks.

2 Literature Review

The field of word embeddings has seen significant advancements over the years, with several seminal works laying the foundation for modern techniques. Some of the key contributions include:

- **Word2Vec**: Introduced by Mikolov et al. [Mikolov et al.(2013a)Mikolov, Chen, Corrado, and Dean, Mikolov et al.(2013b)Mikolov, Sutskever, Chen, Corrado, and Dean], Word2Vec is one of the most influential models for learning word embeddings. It includes two architectures: the Continuous Bag of Words (CBOW) and the Skip-gram model. These models efficiently estimate word representations in vector space, capturing semantic and syntactic similarities.
- **GloVe**: Pennington et al. [Pennington et al.(2014)Pennington, Socher, and Manning] developed the GloVe (Global Vectors for Word Representation) model, which combines global matrix factorization with local context window methods. GloVe constructs a word-word co-occurrence matrix and learns word vectors such that their dot product approximates the logarithm of the co-occurrence probability.
- **BERT**: Devlin et al. [Devlin et al.(2018)Devlin, Chang, Lee, and Toutanova] introduced BERT (Bidirectional Encoder Representations from Transformers), a transformer-based model that generates contextual word embeddings. BERT uses a multi-layer bidirectional transformer encoder to capture the context of words in a sentence, making it highly effective for various NLP tasks.
- **FastText**: Bojanowski et al. [Bojanowski et al.(2017)Bojanowski, Grave, Joulin, and Mikolov] presented FastText, an extension of the Word2Vec model that represents words as a sum of their character n-grams. This approach allows FastText to handle out-of-vocabulary words and morphologically rich languages more effectively.

3 Mathematical Formulation

Let V be the vocabulary size and d be the dimension of the embedding space. A word embedding is a mapping from the vocabulary to a d-dimensional vector space:

$$\mathbf{w} : \{1, 2, \ldots, V\} \to \mathbb{R}^d$$

For a word i in the vocabulary, its embedding vector is denoted as $\mathbf{w}_i \in \mathbb{R}^d$.

3.1 Objective Function

The objective of word embeddings is to learn a representation that maximizes the similarity between words that appear in similar contexts. One common approach is the Skip-gram model with Negative Sampling (SGNS), which aims to maximize the probability of predicting context words given a target word.

The objective function for SGNS is:

$$\mathcal{L} = \sum_{(w,c) \in D} \left[\log \sigma(\mathbf{w}_w \cdot \mathbf{w}_c) + \sum_{c' \in \text{Neg}(w)} \log \sigma(-\mathbf{w}_w \cdot \mathbf{w}_{c'}) \right]$$

where:

- D is the set of (target word, context word) pairs.
- σ is the sigmoid function.
- $\text{Neg}(w)$ is the set of negative samples for the target word w.

3.2 Training

The embedding vectors $\mathbf{w}_i$ are learned by optimizing the objective function using stochastic gradient descent (SGD) or other optimization algorithms. The gradient of the loss function with respect to the embedding vectors is given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_w} = \sum_{(w,c) \in D} \left[(\sigma(\mathbf{w}_w \cdot \mathbf{w}_c) - 1)\mathbf{w}_c + \sum_{c' \in \text{Neg}(w)} (\sigma(-\mathbf{w}_w \cdot \mathbf{w}_{c'}) - 1)\mathbf{w}_{c'} \right]$$

4 Skip-gram Model

The Skip-gram model is a popular method for learning word embeddings. It predicts the context words given a target word. This model was introduced by Mikolov et al. [Mikolov et al.(2013a)Mikolov, Chen, Corrado, and Dea Mikolov et al.(2013b)Mikolov, Sutskever, Chen, Corrado, and Dean].

4.1 Model Architecture

The Skip-gram model uses a neural network with a single hidden layer. The input is a one-hot encoded vector of the target word, and the output is a probability distribution over the context words.

4.2 Training Procedure

The training procedure involves the following steps:

1. Initialize the embedding matrix $\mathbf{W}$ randomly.
2. For each (target word, context word) pair in the training data, compute the dot product of their embedding vectors.
3. Use the sigmoid function to compute the probability of the context word given the target word.
4. Update the embedding vectors using gradient descent to maximize the objective function.

5 Continuous Bag of Words (CBOW) Model

The Continuous Bag of Words (CBOW) model is another popular method for learning word embeddings. It predicts the target word given the context words. This model was also introduced by Mikolov et al. [Mikolov et al.(2013a)Mikolov, Chen, Corrado, and Dean, Mikolov et al.(2013b)Mikolov, Sutskever, Chen, Corrado, and

5.1 Model Architecture

The CBOW model uses a neural network with a single hidden layer. The input is the average of the one-hot encoded vectors of the context words, and the output is a probability distribution over the target words.

5.2 Training Procedure

The training procedure for the CBOW model is similar to the Skip-gram model, but the input is the average of the context word vectors instead of a single target word vector.

6 Global Vectors for Word Representation (GloVe)

Global Vectors for Word Representation (GloVe) is a method that combines the advantages of global matrix factorization and local context window methods. This method was introduced by Pennington et al. [Pennington et al.(2014)Pennington, Socher, and Manning].

6.1 Model Formulation

GloVe constructs a word-word co-occurrence matrix X where X_{ij} represents the number of times word j occurs in the context of word i. The objective is to learn word vectors such that their dot product is equal to the logarithm of the co-occurrence probability:

$$\mathbf{w}_i \cdot \mathbf{w}_j + b_i + b_j = \log X_{ij}$$

where b_i and b_j are bias terms for words i and j.

6.2 Training Procedure

The training procedure involves minimizing the following objective function:

$$\mathcal{L} = \sum_{i,j=1}^{V} f(X_{ij})(\mathbf{w}_i \cdot \mathbf{w}_j + b_i + b_j - \log X_{ij})^2$$

where $f(X_{ij})$ is a weighting function that gives less weight to rare and frequent co-occurrences.

7 Contextual Word Embeddings

Contextual word embeddings, such as those generated by models like BERT, capture the context-dependent meaning of words. Unlike static embeddings, contextual embeddings generate different vectors for the same word depending on its context.

7.1 BERT Model

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that generates contextual word embeddings. It uses a multi-layer bidirectional transformer encoder to capture the context of words in a sentence.

7.1.1 Model Architecture

BERT consists of multiple layers of transformer encoders, each with a self-attention mechanism and a feed-forward neural network. The input to BERT is a sequence of token embeddings, position embeddings, and segment embeddings.

7.1.2 Training Procedure

BERT is trained using two objectives:

1. Masked Language Modeling (MLM): Randomly mask some of the tokens in the input and predict the original tokens.
2. Next Sentence Prediction (NSP): Given two sentences, predict whether the second sentence is the actual next sentence.

8 Subword Embeddings

Subword embeddings, such as those generated by models like FastText, capture the meaning of subword units (e.g., character n-grams). This is particularly useful for handling out-of-vocabulary words and morphologically rich languages.

8.1 FastText Model

FastText is an extension of the Word2Vec model that represents words as a sum of their character n-grams. This allows it to capture the meaning of subword units and handle out-of-vocabulary words.

8.1.1 Model Architecture

FastText uses a bag-of-character-ngrams representation for words. The embedding of a word is the sum of the embeddings of its character n-grams.

8.1.2 Training Procedure

The training procedure for FastText is similar to Word2Vec, but it uses the bag-of-character-ngrams representation for words.

9 From Word Embeddings to Meaning Space

Word embeddings serve as the foundation for representing the meaning of words in a continuous vector space. However, to fully capture the semantic meaning of text, we need to move beyond individual word embeddings and consider the meaning space, which encompasses the relationships between words, phrases, and entire sentences.

9.1 Compositional Semantics

Compositional semantics refers to the idea that the meaning of a complex expression is determined by the meanings of its constituent parts and the rules used to combine them. In the context of word embeddings, this means that the meaning of a sentence or a phrase can be derived from the embeddings of its constituent words.

9.1.1 Vector Space Models

Vector space models represent the meaning of text as vectors in a high-dimensional space. By combining the embeddings of individual words, we can create representations for phrases and sentences. This can be achieved through simple operations such as addition or more complex operations such as recurrent neural networks (RNNs) or transformers.

Let's denote the embedding of a word w as $\mathbf{w}$. For a sentence S consisting of words $w_1, w_2, \ldots, w_n$, a simple way to represent the sentence is by averaging the word embeddings:

$$\mathbf{S} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{w}_i$$

However, this simple averaging does not capture the order of words or the syntactic structure of the sentence. More advanced methods, such as RNNs and transformers, can capture these nuances.

9.1.2 Recurrent Neural Networks (RNNs)

RNNs are a type of neural network that can process sequences of data, making them well-suited for NLP tasks. RNNs can capture the temporal dependencies between words in a sentence, allowing them to generate contextual embeddings that reflect the meaning of the entire sentence.

Given a sequence of word embeddings $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$, an RNN processes these embeddings sequentially. The hidden state $\mathbf{h}_t$ at time step t is updated as follows:

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{w}_t + \mathbf{b})$$

where $\mathbf{W}_h$ and $\mathbf{W}_x$ are weight matrices, and $\mathbf{b}$ is a bias vector. The final hidden state $\mathbf{h}_n$ can be used as the representation of the sentence.

9.1.3 Long Short-Term Memory (LSTM) Networks

LSTM networks are a type of RNN that can capture long-term dependencies in sequences. LSTMs introduce memory cells that can maintain information over long periods, making them more effective for tasks that require understanding the context of a sentence.

The LSTM cell has three gates: the input gate $\mathbf{i}_t$, the forget gate $\mathbf{f}_t$, and the output gate $\mathbf{o}_t$. The cell state $\mathbf{c}_t$ and the hidden state $\mathbf{h}_t$ are updated as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{w}_t + \mathbf{b}_i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{w}_t + \mathbf{b}_f)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{w}_t + \mathbf{b}_o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{w}_t + \mathbf{b}_c)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where σ is the sigmoid function, $\odot$ denotes element-wise multiplication, and $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c, \mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_o, \mathbf{U}_c$ are weight matrices.

9.1.4 Gated Recurrent Units (GRUs)

GRUs are another type of RNN that can capture long-term dependencies. GRUs have a simpler architecture than LSTMs but can still capture important temporal dependencies.

The GRU cell has two gates: the update gate $\mathbf{z}_t$ and the reset gate $\mathbf{r}_t$. The hidden state $\mathbf{h}_t$ is updated as follows:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{w}_t + \mathbf{b}_z)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{w}_t + \mathbf{b}_r)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}_h \mathbf{w}_t + \mathbf{b}_h)$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

where $\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}_h, \mathbf{U}_z, \mathbf{U}_r, \mathbf{U}_h$ are weight matrices, and $\mathbf{b}_z, \mathbf{b}_r, \mathbf{b}_h$ are bias vectors.

9.1.5 Transformers

Transformers are a more recent architecture that has achieved state-of-the-art performance on a wide range of NLP tasks. Transformers use self-attention mechanisms to capture the relationships between all words in a sentence, allowing them to generate highly contextual embeddings.

The self-attention mechanism computes a weighted sum of the input embeddings, where the weights are determined by the similarity between the embeddings. For a sequence of word embeddings $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n]$, the self-attention output $\mathbf{Z}$ is computed as:

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

where $\mathbf{Q} = \mathbf{W}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{W}\mathbf{W}_K$, and $\mathbf{V} = \mathbf{W}\mathbf{W}_V$ are the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors.

9.2 Semantic Similarity

Semantic similarity measures the degree to which two pieces of text have the same meaning. Word embeddings can be used to compute semantic similarity by measuring the distance between the embeddings of words, phrases, or sentences.

9.2.1 Cosine Similarity

Cosine similarity is a common measure of semantic similarity that computes the cosine of the angle between two vectors. For word embeddings, cosine similarity can be used to measure the similarity between the embeddings of two words, phrases, or sentences.

The cosine similarity between two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by:

$$\text{cosine\_similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}$$

where $\mathbf{a} \cdot \mathbf{b}$ is the dot product of $\mathbf{a}$ and $\mathbf{b}$, and $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ are the magnitudes of $\mathbf{a}$ and $\mathbf{b}$, respectively.

9.2.2 Euclidean Distance

Euclidean distance is another measure of semantic similarity that computes the straight-line distance between two points in a vector space. For word embeddings, Euclidean distance can be used to measure the similarity between the embeddings of two words, phrases, or sentences.

The Euclidean distance between two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by:

$$\text{euclidean\_distance}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}$$

where d is the dimension of the vectors, and a_i and b_i are the components of $\mathbf{a}$ and $\mathbf{b}$, respectively.

9.2.3 Manhattan Distance

Manhattan distance, also known as L1 distance, is another measure of semantic similarity that computes the sum of the absolute differences between the components of two vectors. For word embeddings, Manhattan distance can be used to measure the similarity between the embeddings of two words, phrases, or sentences.

The Manhattan distance between two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by:

$$\text{manhattan\_distance}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{d} |a_i - b_i|$$

where d is the dimension of the vectors, and a_i and b_i are the components of $\mathbf{a}$ and $\mathbf{b}$, respectively.

9.2.4 Minkowski Distance

Minkowski distance is a generalization of Euclidean and Manhattan distances. It computes the p-th root of the sum of the p-th powers of the differences between the components of two vectors. For word embeddings, Minkowski distance can be used to measure the similarity between the embeddings of two words, phrases, or sentences.

The Minkowski distance between two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by:

$$\text{minkowski\_distance}(\mathbf{a}, \mathbf{b}, p) = \left(\sum_{i=1}^{d} |a_i - b_i|^p \right)^{\frac{1}{p}}$$

where d is the dimension of the vectors, a_i and b_i are the components of $\mathbf{a}$ and $\mathbf{b}$, respectively, and p is a parameter that determines the type of distance (e.g., $p = 2$ for Euclidean distance, $p = 1$ for Manhattan distance).

9.3 Applications of Meaning Space

The meaning space represented by word embeddings and their compositions has numerous applications in NLP, including:

- **Text classification**: Classifying text into predefined categories based on its meaning. Text classification is a fundamental task in NLP, where the goal is to assign predefined categories or labels to textual data. Word embeddings can be used to represent the text in a continuous vector space, which can then be fed into a classifier to predict the category.
- **Named entity recognition**: Identifying and classifying named entities in text. Named entity recognition (NER) involves identifying and categorizing named entities such as people, organizations, and locations in text. Word embeddings can help capture the contextual information needed to accurately identify and classify these entities.
- **Machine translation**: Translating text from one language to another while preserving its meaning. Machine translation involves converting text from one language to another. Word embeddings can be used to represent the source and target languages in a shared vector space, facilitating the translation process.
- **Sentiment analysis**: Determining the sentiment or emotional tone of a piece of text. Sentiment analysis involves determining the emotional tone or sentiment expressed in a piece of text. Word embeddings can capture the semantic nuances required to accurately classify the sentiment of the text.
- **Information retrieval**: Retrieving relevant information based on the meaning of a query. Information retrieval involves finding and ranking relevant documents or information based on a query. Word embeddings can be used to represent the query and documents in a vector space, allowing for effective similarity matching.
- **Question answering**: Answering questions based on the meaning of a piece of text. Question answering involves providing accurate answers to questions based on a given context or knowledge base. Word embeddings can help capture the semantic relationships between the question and the context, enabling accurate answer retrieval.
- **Text summarization**: Generating a concise summary of a longer piece of text while preserving its meaning. Text summarization involves creating a shorter version of a document while retaining its key information. Word embeddings can be used to represent the text and identify important sentences or phrases for inclusion in the summary.
- **Paraphrase identification**: Identifying whether two pieces of text convey the same meaning. Paraphrase identification involves determining whether two pieces of text express the same meaning. Word embeddings can capture the semantic similarities between the texts, aiding in accurate identification of paraphrases.
- **Coreference resolution**: Identifying and resolving references to the same entity within a text. Coreference resolution involves identifying and linking references to the same entity within a text. Word embeddings can help capture the contextual information needed to accurately resolve coreferences.

10 Evaluation of Word Embeddings

Evaluating word embeddings is crucial to ensure their effectiveness. Common evaluation methods include:

- **Intrinsic evaluations**: Measure the quality of the embeddings directly, such as word similarity tasks. Intrinsic evaluations assess the quality of word embeddings by directly measuring their performance on specific linguistic tasks. These tasks include word similarity, word analogy, and synonym detection. The goal is to evaluate how well the embeddings capture semantic and syntactic relationships between words.

- **Extrinsic evaluations**: Measure the performance of the embeddings on downstream NLP tasks. Extrinsic evaluations assess the quality of word embeddings by measuring their performance on downstream NLP tasks. These tasks include text classification, named entity recognition, machine translation, and sentiment analysis. The goal is to evaluate how well the embeddings contribute to the overall performance of NLP systems.
- **Qualitative evaluations**: Involve human judgment to assess the quality of the embeddings. Qualitative evaluations involve human judgment to assess the quality of word embeddings. This can include tasks such as manually inspecting the nearest neighbors of words in the embedding space to ensure that they are semantically similar. The goal is to evaluate the embeddings based on human intuition and understanding.

11 Advanced Topics

11.1 Multilingual Word Embeddings

Multilingual word embeddings capture the semantic similarities between words across different languages. This is useful for tasks such as machine translation and cross-lingual information retrieval.

11.1.1 Model Formulation

Multilingual word embeddings can be learned using bilingual dictionaries or parallel corpora. The objective is to align the embedding spaces of different languages such that semantically similar words have similar embeddings. This can be achieved using techniques such as canonical correlation analysis (CCA) or adversarial training.

11.1.2 Training Procedure

The training procedure involves minimizing the distance between the embeddings of semantically similar words across different languages. This can be achieved using techniques such as canonical correlation analysis (CCA) or adversarial training. The goal is to ensure that the embeddings of semantically similar words in different languages are close to each other in the embedding space.

11.1.3 Applications

Multilingual word embeddings have numerous applications, including machine translation, cross-lingual information retrieval, and multilingual text classification. They enable NLP systems to handle multiple languages effectively by capturing the semantic relationships between words across different languages.

11.2 Knowledge Graph Embeddings

Knowledge graph embeddings represent entities and relations in a knowledge graph as vectors in a continuous vector space. This is useful for tasks such as link prediction and entity resolution.

11.2.1 Model Formulation

Knowledge graph embeddings can be learned using techniques such as TransE, DistMult, or ComplEx. The objective is to learn embeddings that capture the structural properties of the knowledge graph. These techniques model the relationships between entities and relations in the knowledge graph and learn embeddings that preserve these relationships.

11.2.2 Training Procedure

The training procedure involves minimizing a loss function that measures the discrepancy between the predicted and actual relations in the knowledge graph. This can be achieved using techniques such as negative sampling or margin ranking. The goal is to ensure that the embeddings accurately capture the structural properties of the knowledge graph.

11.2.3 Applications

Knowledge graph embeddings have numerous applications, including link prediction, entity resolution, and knowledge graph completion. They enable NLP systems to effectively represent and reason about the relationships between entities and relations in a knowledge graph.

11.3 Subword Embeddings

Subword embeddings, such as those generated by models like FastText, capture the meaning of subword units (e.g., character n-grams). This is particularly useful for handling out-of-vocabulary words and morphologically rich languages.

11.3.1 Model Formulation

Subword embeddings are generated using models such as FastText, which represents words as a sum of their character n-grams. This approach allows the model to capture the meaning of subword units and handle out-of-vocabulary words. The embedding of a word is the sum of the embeddings of its character n-grams.

11.3.2 Training Procedure

The training procedure for subword embeddings is similar to that of word embeddings, but it uses the bag-of-character-ngrams representation for words. The model is trained to predict the context words given a target word, and the embeddings are updated using gradient descent.

11.3.3 Applications

Subword embeddings have numerous applications, including text classification, named entity recognition, machine translation, and sentiment analysis. They enable NLP systems to handle out-of-vocabulary words and morphologically rich languages more effectively.

12 Conclusion

The mathematical study of word embeddings reveals how meaning can be systematically represented in continuous vector spaces. These meaning spaces capture semantic relationships through geometric operations, suggesting fundamental connections between linguistic structure and spatial geometry. The progression from static to contextual embeddings, and from monolingual to multilingual spaces, demonstrates how the mathematical framework can accommodate increasingly sophisticated models of meaning. The success of embedding-based meaning spaces in natural language tasks points to deep connections between geometric properties and semantic relationships. While current approaches have achieved remarkable results, important questions remain about the nature of semantic composition in vector spaces and the relationship between linguistic and geometric distance. Future work should focus on developing more rigorous theories of how meaning spaces can represent complex semantic phenomena and how different embedding approaches capture complementary aspects of language understanding. The mathematical foundations presented here provide a framework for exploring these questions while advancing our understanding of how to represent meaning in computational systems.

References

[Bojanowski et al.(2017)Bojanowski, Grave, Joulin, and Mikolov] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[Devlin et al.(2018)Devlin, Chang, Lee, and Toutanova] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Mikolov et al.(2013a)Mikolov, Chen, Corrado, and Dean] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

[Mikolov et al.(2013b)Mikolov, Sutskever, Chen, Corrado, and Dean] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013b.

[Pennington et al.(2014)Pennington, Socher, and Manning] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014.

13 Appendix

13.1 Example Code for Training Word Embeddings

Here is an example code snippet for training word embeddings using the Gensim library in Python:

```
from gensim.models import Word2Vec
# Sample sentences
sentences = [['this', 'is', 'the', 'first', 'sentence'],
    ['this', 'is', 'the', 'second', 'sentence'],
    ['yet', 'another', 'sentence']]
# Train the Word2Vec model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
# Get the vector for a word
vector = model.wv['sentence']
```

13.2 Algorithm for Training Word2Vec

Here is a pseudocode algorithm for training the Word2Vec model:

Algorithm 1 Training Word2Vec Model

```
procedure TrainWord2Vec(corpus, vector_size, window, min_count, workers)
    Initialize embedding matrix W randomly
    for each sentence in corpus do
        for each (target word, context word) pair in sentence do
            Compute dot product of embedding vectors
            Compute probability using sigmoid function
            Update embedding vectors using gradient descent
        end for
    end for
    return embedding matrix W
end procedure
```

13.3 Mathematical Details of Word2Vec Training

13.3.1 Objective Function

The objective function for the Skip-gram model with Negative Sampling (SGNS) is given by:

$$\mathcal{L} = \sum_{(w,c) \in D} \left[\log \sigma \left(\mathbf{w}_w \cdot \mathbf{w}_c \right) + \sum_{c' \in \text{Neg}(w)} \log \sigma \left(-\mathbf{w}_w \cdot \mathbf{w}_{c'} \right) \right]$$

where:

- D is the set of (target word, context word) pairs.
- σ is the sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$.
- $\text{Neg}(w)$ is the set of negative samples for the target word w.

13.3.2 Gradient Calculation

The gradient of the loss function with respect to the embedding vectors is given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_w} = \sum_{(w,c)\in D} \left[\left(\sigma\left(\mathbf{w}_w \cdot \mathbf{w}_c\right) - 1\right) \mathbf{w}_c + \sum_{c' \in \text{Neg}(w)} \left(\sigma\left(-\mathbf{w}_w \cdot \mathbf{w}_{c'}\right) - 1\right) \mathbf{w}_{c'} \right]$$

13.3.3 Update Rule

The embedding vectors are updated using the gradient descent rule:

$$\mathbf{w}_w \leftarrow \mathbf{w}_w - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_w}$$

where η is the learning rate.

13.3.4 Negative Sampling

Negative sampling is used to approximate the softmax function, which is computationally expensive. The negative samples are drawn from a noise distribution, typically the unigram distribution raised to the power of $\frac{3}{4}$.

13.3.5 Context Window

The context window defines the range of context words to consider for each target word. For a context window size of c, the context words for a target word at position t are the words in the range $[t-c, t+c]$.

13.3.6 Subsampling

Subsampling is used to reduce the frequency of common words in the training data. The probability of keeping a word is given by:

$$P\left(w_i\right) = \left(\sqrt{\frac{z\left(w_i\right)}{0.001}} + 1\right) \frac{0.001}{z\left(w_i\right)}$$

where $z\left(w_i\right)$ is the frequency of word w_i.