6-1-2023

GraphWords: Enhancing Word Vectors using Graphs

Mrulay Sureshbhai Mistry
University of Windsor

GraphWords: Enhancing Word Vectors using Graphs

By

Mrulay Mistry

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2023

GraphWords: Enhancing Word Vectors using Graphs

by

Mrulay Mistry

APPROVED BY:

R. Nakhaie
Department of Sociology and Criminology

H. Fani
School of Computer Science

Z. Kobti, Advisor
School of Computer Science

April 24, 2023

DECLARATION OF ORIGINALITY

ABSTRACT

In the domain of Natural Language Processing (NLP) the representation of words according to their distribution in the vector form is a crucial task. In the representation space, when words that are similar to each other according to human interpretation are placed closer to each other, a notable increase can be observed in the performance and accuracy of NLP tasks. Previous word embedding methods put emphasis on passing the word tokens in an iterative manner through a Neural Language model to capture the context of the words. These methods can capture word relatedness, but only within the given context length. In this thesis, we introduce a method to represent the words in the form of a graph so that their first-order and second-order proximity are preserved and the relatedness of a word can be captured in a better manner. This graph is then subjected to a vertex (node) embedding method to generate their embedding. After experimenting with the proposed method on multiple text corpora, our findings indicate that this method of word representation outperforms the traditional word-embedding methods by more than 4% in multiple intrinsic tasks and extrinsic tasks.

DEDICATION

I would like to dedicate this thesis to my mom for her incredible love and support. Because I believe that she is the real backbone of our family, this is to appreciate her selfless hard work and efforts toward the family.

Furthermore, I dedicate it to my dad to raise me like a son and give me wings to fly. To my grandfather, for always trusting me and supporting me in my hard times, without his encouragement, nothing would have been easy. And to my entire family for their unconditional affection toward me.

I would also like to dedicate this to my dearest friend, Shriharshini Ganapuram for being a pillar of strength by my side and being my motivation to perform this research. She is the reason I chose to conduct this research and for that, I give her my unconditional and unquestioned respect and admiration.

ACKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervisor Dr. Ziad Kobti, whose input helped me immensely. With his input, I was able to look at my research with a different perspective and a more critical eye.

Secondly, I would like to express my gratitude to my thesis committee members for their beneficial advice and suggestions for my thesis.

I would also like to thank my friends Ali Hassan, Regina Khalil and Lama Khalil for always encouraging and supporting me.

I humbly extend my thanks to the School of Computer Science and all concerned people who helped me in this regard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

NLP	Natural Language Processing
LSA	Latent Semantic Analysis
POS	Part of Speech
NER	Named Entity Recognition
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
CBOW	Continuous Bag of Words
GCN	Graph Convolutional Neural Network
SDNE	Structural Deep Neural Embedding
GGSNN	Gated Graph Sequence Neural Networks
MLP	Multilayer Perceptron
GNN	Graph Neural Network
NBNE	Neighbor Based Node Embeddings
QA	Question Answering
KG	Knowledge Graph
RQ	Research Question
TP	True Positive
FN	False Negative
FP	False Positive
TN	True Negative

CHAPTER 1

Introduction

1.1 Distributional Representation of Words

In Linguistics, the context of a word is mainly responsible for generating its meaning. Consequently, words that frequently occur near each other in a similar semantic space tend to have similar meanings. This leads to the conclusion that the resemblance of words is comparable to how these words are distributed. This hypothesis is known as **Distributional Hypothesis**[29]. Representing words based on their contextual features then became one of the prime focuses of linguistics. Later, with the advent of Machine Learning techniques and the increase in NLP research, more focus was placed on the research of Vector Semantics.

1.2 Vector Semantics and Embedding

Vector Semantics build on the previously introduced Distributional Hypothesis. They learn the representation of words in the form of multi-dimensional vectors based on the distribution of words in texts. These embeddings give computers the ability to understand and process human language in the form of text. For an NLP task, the words must be represented numerically for the computers to understand and process them [55]. Initial methods like [63] and [35] used indexing methods for the word representation where a number would represent a word. While these were the first steps for Computational Linguistics, they could not achieve good results due to their inability to capture context and similarity in the text. These would prove to be inefficient with the increase in vocabulary size.

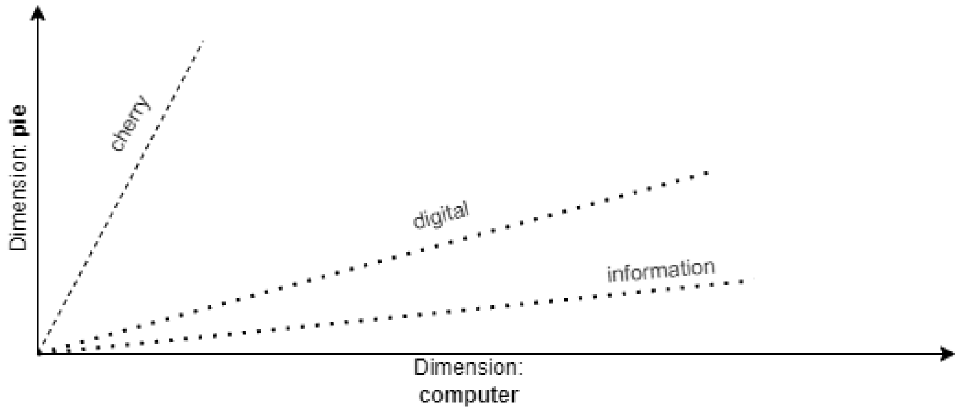

Fig. 1.2.1: An example of how word embeddings work and how they can quantify similarities between words. In this example, we take a look at words within the *(computer, pie)* axes. The words *digital* and *information* are more relevant to *computer* so their vectors are more inclined towards it. While the vector for *cherry* is farther away from *computer* and closer to *pie.*[36]

Representing words just by a single number (indexing method) would not yield good answers in problems like text classification [40] or text generation [34]; the reason being that the correct syntactic meaning of the word cannot be justified as a single index number. To alleviate this, they are represented as vectors, putting them in a syntactic space where each word is defined by its syntactic features.

1.3 Thesis Motivation

1.3.1 The Problem with Conventional Word Vectors

In NLP, two words are deemed to be similar if they often appear near each other. The problem with this notion of similarity is that some words might appear near each other very frequently, but they still won't be similar. For example, take the words 'coffee' and 'cup' in the following examples:

1. *Can I have a **cup** of **coffee**?*
2. *I am going to have a **cup** of **coffee**.*

These phrases often occur in a text corpus, especially in a dialogue corpus. These words appear near each other frequently because they co-participate in the event that happens every day, but that does not directly translate to their similarity. Even though these words belong to the same semantic scope, they cannot be considered similar. Words can be associated with each other in more ways than one way, and this kind of association between words is known as **"relatedness"**[6]. On the other hand, **word similarity** aims to associate words with similar semantic meanings together.[32]

Further, in a lot of cases, it occurs that only domain-specific words are used in a corpus. For example, the word ***Sodium Chloride (NaCl)*** is the scientific word for table salt[70]. In many scientific journals and texts, only Sodium Chloride or NaCl might be used instead of the common utterance of salt/table salt. This would mean that even though there is a similarity in these two terms because they do not occur near to each other in a particular text, these words would be deemed non-similar to each other. Conventional word embedding methods only take into consideration the words that are in the corpus and do not have a way to scale across multiple datasets unless all the datasets are aggregated into one and then the embedding methods are trained on.

There are two types of Word Embedding methods[72]. (i) Static Word Embeddings and (ii) Contextual Word Embeddings. The static methods (for example Word2Vec[52], GloVe[56], etc.) generally build a huge global corpus and produce continuous embeddings based on them. While on the other hand, contextual methods like BERT[14], XLNet[78], etc. learn the semantics of the words based on the previous context. In this paper, we introduce a Static Vector Embedding method that aims to overcome the shortcomings of previous static embedding methods.

Vector-based word models, including both, neural models[52, 56] and traditional word and document indexing methods[13, 43] can capture the context of the words, and that leads to capturing word relatedness; but they are limited by their corpus and cannot scale unless trained further on a bigger dataset.

1.4 Thesis Statement

1.4.1 Graphs and their Usefulness

Graphs are mathematical data structures that contain a set of nodes/vertexes V and a set of edges E. We can denote the graph as $G(V, E)$. The V contains a number of nodes in the graph, i.e.

$$V = \{n_1, n_2, ...n_i\} \tag{1.1}$$

While in the edge set, each edge e_{ij} denotes a connection between node n_i and n_j. These edges can have various properties such as weight and direction to signify some specific characteristic of the graph.

Using graphs as a data structure increases the scope of understanding the data points in many problems. They provide a comprehensive understanding of how each data point is interconnected with other data points which leads to a better quantitative understanding of the problem and less complex solutions can be developed for unconventional problems.

In the context of our research, each word in a Text Corpus can be represented as a node. Depending on their usage and similarity, edges can be created between the words with each edge having some weight. This weight of the edge can determine how similar two words are. Thus, the textual data can be converted into a graph representation that signifies the connections (or similarities) between the words.

1.4.2 Enhancement using Graph Embeddings

A graph data structure provides two key characteristics of the neighbors of each node. The first-order and the second-order proximity. [6.2.1]

Thus, graphs allow us to represent relational data in a structured way. With their ability to capture multi-level proximity, they are known to capture relations and associations [59]. This reason leads us to the following hypothesis:

Hypothesis: Since Graphs are known to perform better in representing entities with contextual relations, using Graph Embedding methods on a relational graph with every word represented as a node and each node connected to the words in its context would yield better results for the tasks in NLP involving relations between words.

1.4.3 Objectives

The mathematical problem formulation of the problem statement is as follows:

Task 1: Given a Text Corpus C containing n words, find a graphical representation G where each word is structured as a node and each node is connected to the words in its context buffer of size C_n.

$$V_i = d_1^i, d_1^i, d_1^i, d_1^i ... d_m^i \tag{1.2}$$

where m is the size of vector representation, d_i is the vector dimension of i^{th} word

Task 2: For each pair of Word Vectors $\{V_i, V_{i+1}\}$, justify a reasoning factor R_i^{i+1} (Edge between the two nodes) to explain the positioning of the word vectors and their relations to the neighboring words.

1.5 Thesis Organization

The first section of this thesis motivates why a new method of vector embedding is required. It lays out an apparent problem and introduces a new method that can potentially overcome the shortcomings of all previously mentioned methods.

The second section of the thesis gives the literature review of the selected problem. It begins with an explanation of how the idea of word embedding was introduced and how it was improved over the years and eventually moves on to introduce current state-of-the-art methods which are widely used. This section also covers the selected baselines in detail.

The following section covers all the underlying machine-learning methods used in this thesis. As a part of the proposed method, we use SDNE[27], Node2Vec[28], and

GGSNN [47] methods to test out different natures of Graph Embedding methods to get the best results.

The fourth section is the proposed method. It introduces a three-phased approach that we propose in this thesis. It involved graph creation from text, its enhancement, and embedding generation based on the nodes of the graph.

The fifth section elaborates on the experiments performed on the proposed method for its evaluation. It also gives details about the hardware that was used to perform the tests and the results obtained. Based on the results, this section also explains the behavior of the method on different tests across multiple datasets.

Finally, the last section concludes the discussion and also gives some ideas on how this thesis can be taken forward. It establishes an idea of a global pre-trained dataset that can be prepared using the proposed method.

CHAPTER 2

Related Works

The idea of an NLP system being able to capture the meaning of words is dependent on the hypothesis proposed by L. Wittgenstein [75] and Z. Harris [29]. Their hypothesis states that words are distributed in multiple environments which are composed of an existing cluster of other co-occurring words. They strengthen their hypothesis further by stating that a language element's meaning can be related to its distribution.

There are a number of factors on which the words embeddings are dependent upon[36]:

Wordforms/Lemmas: To eliminate the repetition of different forms of the same word, the words need to be converted to their **lemmas** or **citation forms**. A lemma is a shortened parent-form of a word. For example, *sing* is the lemma for the words *sing, sang, and sung.*

Synonymy: A word can be considered as a synonym to another word if the two words can be used in place of each other. Even after the substitution, the sense of the overall text does not change.

But it might happen that words sometimes are not synonyms but they do carry the same sense. For example the words *cup* and *mug.* They cannot be considered synonyms, but they carry the same sense of a container. This is known as the principle of contrast [8]. This gives rise to the concept of **Word Similarity**. This understanding of the similarity between words is very important in semantic and various NLP tasks as well.

In 2003, Bengio et al. [4] introduced the first neural language model. In their paper, they mapped each unique word with a number (indexes). These indexes are then passed to a feed-forward network with a hidden layer to predict the next word.

2.1 Early Vector Representations of Words

For a word representation to produce good results in NLP tasks, it should be able to capture both the semantic and syntactic aspects of the words [66]. Before the introduction of Neural word embedding models, the words were simply represented by their strings. This meant that each word had its own identity and that gave rise to the indexing method. While it was the simplest approach to represent words as quantifiable numbers, it did not prove to be a very good way to represent words.

2.1.1 Indexing Methods for Word Embedding

One of the early mathematical models that generated features of words was the Latent Semantic Analysis (LSA) model[41]. This method followed the Distributional Hypothesis where a matrix is formed by calculating the frequency of each word in different documents. This matrix is then reduced by Singular Valued Decomposition (SVD) method. These matrices are then used against matrices of other documents to find similarities between them.

Other models involved words being represented in the Vector Space according to their frequencies in documents. These methods involved forming a term-document matrix or considering global co-occurrences of words in a word-context matrix [13, 43]. These initial methods performed well in capturing the syntactic aspect of the words but failed to capture the semantic meanings of the word given their context.

2.2 Unsupervised and Semi-Supervised Embedding Methods

The task of generating an embedding does not have a target variable in the dataset. I.e. we cannot hand-craft the embeddings of vast dimensions for each word in multiple documents for a machine-learning model to learn. Hence, this task cannot be supervised.

Methods proposed by Collobert and Weston in [10] and [11] are one of the first semi-supervised vector representation models. In their paper, they used methods like Part-of-Speech (POS) tagging, Named Entity Recognition (NER) and semantic labeling to generate some features which were then passed to a Convolutional Neural Network (CNN) to generate their embedding.

2.2.1 Word2Vec

Neural Word Embedding models are the unsupervised methods of efficiently and accurately calculating the vector representation of words along with capturing their semantic and syntactic assets. Word2Vec, an architecture proposed by Mikolov et al. [52] is the first neural embedding model that was able to achieve this goal. The Word2Vec has two variants: Continuous Bag-of-Words (CBOW) model and the Skip-gram model.

They function on the same principle where the neural network performs a fake task. In the CBOW model, the fake task is to predict the word in the center given the words in its context, while the Skip-Gram model tries to predict the context words given the word in the center. In both models, the weights of the hidden layer are then saved as the vector representation for the subject word.

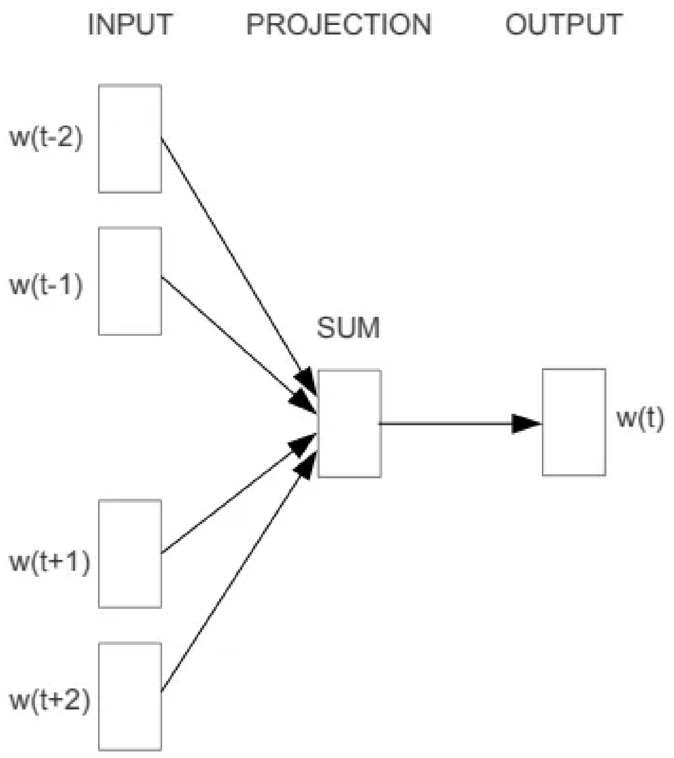

Fig. 2.2.1: The CBOW architecture as proposed in the Word2Vec model. Here, the goal is to predict a target word based on its context words of the desired window size. In this case, the window size is 2.

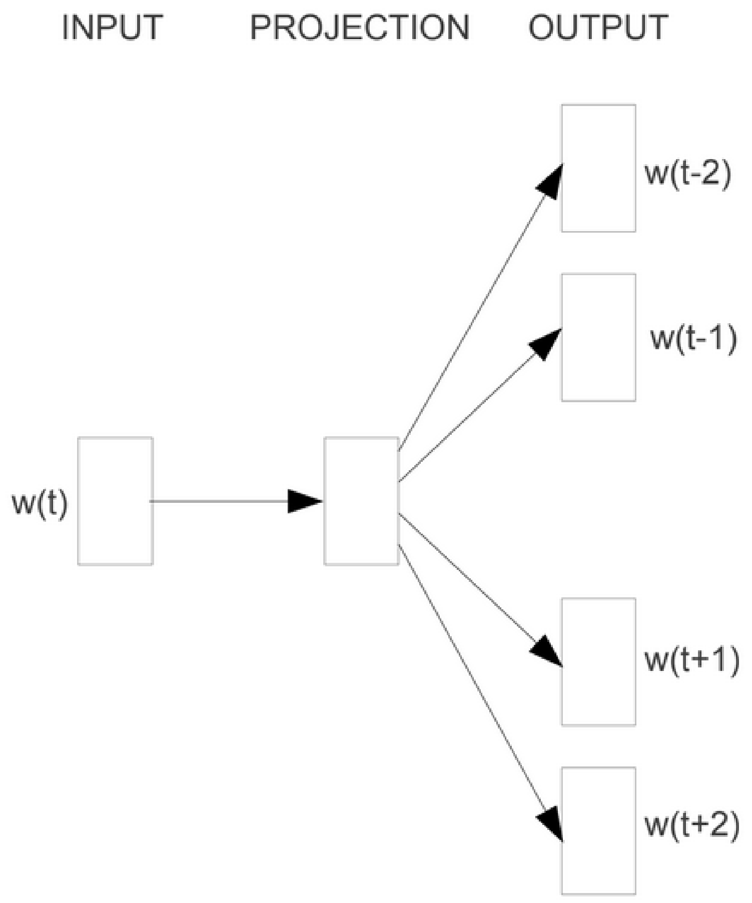

Fig. 2.2.2: The Skip-Gram architecture as shown in the picture takes in the subject word as an argument. Based on the subject word, its context words get predicted in the window size (2 in the figure).

2.2.2 GloVe: Global Vectors for Word Representation

GloVe is one of the state-of-the-art methods proposed by the researchers at Stanford[56]. This algorithm constructs a global co-occurrence matrix that has the probabilities of different pairs of words appearing near to each other. Based on these probabilities, the embedding of each word in the Corpus gets generated. But the downside of this approach is that it takes a lot of time to calculate the vectors and hence, mostly re-used as a pre-trained word embedding.

2.2.3 Other Approaches

A lot of improvements have also been proposed later down the line, such as Dict2Vec [68], where the authors used dictionary information to enhance the word embeddings by forming strong pairs and weak pairs between words. The approach put forward by Ling et al. [49] aims to better the Word2Vec model [52] by using an attention model to solve the problem of the importance of context words in the original model.

2.3 Use of Graphs in Word Embeddings

While extensive work has been done over the years to improve Word Embedding, the use of Graphs in this area has been very limited. To improve Word Embedding matrices, M. Faruqui et al. [19] in their paper introduces a method known as retrofitting where they represented each word as a node and each relation between the words as an edge. These nodes and edge information were then used with the pre-calculated Word Embedding to bring the related words closer to each other in the Vector Space. Their paper was able to preserve similarity in the pre-calculated/pre-trained word embeddings like Skip-Gram, CBOW [52] and GloVe [56].

The very first widely known usage of Graph Neural Networks based Word Embeddings was given by L. Fang et al. [18]. They used an Ensemble Method to combine the information gained through a Word-Graph like Wordnet synsets [54] along with the traditional Neural Word Embedding models to leverage the merits of both approaches.

2.3.1 SynGCN

In the same year, S. Vashishth et al. [69] proposed the SynGCN and SemGCN frameworks. Their approach involves the transformation of the text corpus to graphs by using dependency parsing to preserve the contextual and positional information. The nodes of the graph are then initialized according to the pre-trained vectors from Word2Vec[52] or GloVe[56]. This graph is then subjected to a Graph Convolutional

Neural Network (GCN) which it generates embedding for each node.

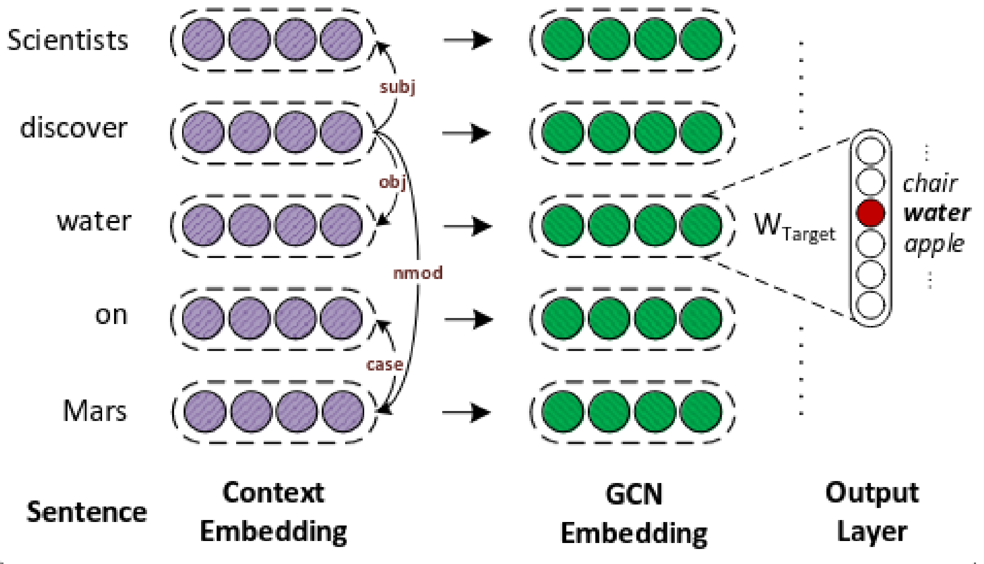

Fig. 2.3.1: The architecture of SynGCN

In the above figure, the pre-trained embeddings of words are converted to a graph-structured based on its dependency tree. This graph structure is then subjected to a GCN to encode and decode (and generate) the new embeddings. The downside to this algorithm is that we need to have pre-trained embeddings to further enhance the embeddings using the GCN. This means more expenses in terms of computation and time.

This approach converts the one-dimensional text data to a graph and uses GCNs to generate embeddings of the words; thus, is along the lines of the scope of our research.

CHAPTER 3

Machine Learning Techniques

This section summarizes all the machine learning methods we use in this research.

3.1 Node Embedding in Graphs

We define graphs as $G(V, E)$ where V is the set of vertices and E is the set of edges that connect two vertices or nodes. In a lot of problems that involve representing the data in the form of a graph, analysis of the graph is one of the most important things to gain insights about the data and give probable solutions to the problem[76].

Graph analysis can prove to be very helpful but at the same time, their computation cost can also become very high when the size of the graph becomes too high[67]. Thus, the need arises to convert the graph from a high-dimensional space into a low-dimensional space. For this purpose, Graph embedding is used. Graph embeddings are much similar to the vector embeddings of words in the NLP domain; they are dense and continuous vector spaces that preserve the properties of the structure of the graph.

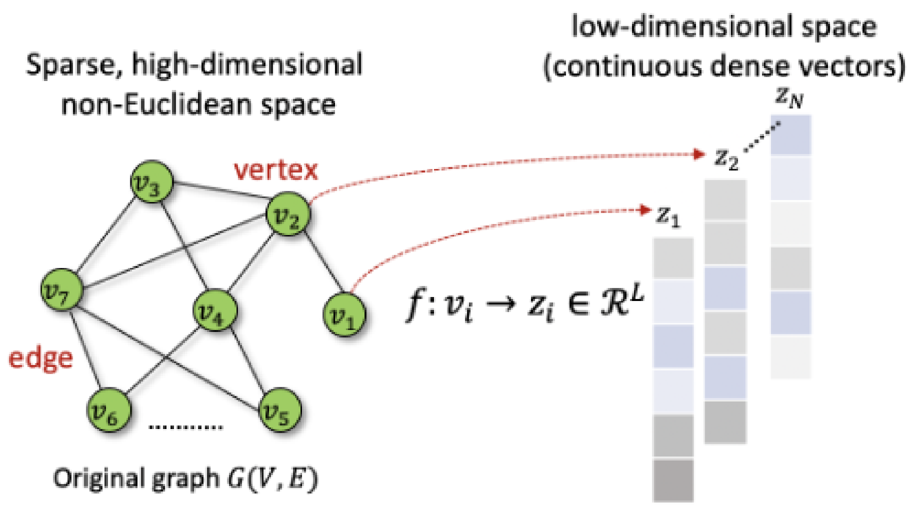
Sparse, high-dimensional non-Euclidean space
low-dimensional space (continuous dense vectors)
vertex
edge
$f: v_i \rightarrow z_i \in \mathcal{R}^L$
z_1
z_2
z_N
v_1
v_2
v_3
v_4
v_5
v_6
v_7
Original graph $G(V, E)$

Fig. 3.1.1: An intuition on Graph Embeddings. In the figure, the nodes from the high-level graph are converted to low-level vectors.[76]

Typically, graph embedding methods belong to one of the three major categories[12]:

- *Matrix Factorization Based Methods*
- *Random-Walk based Methods*
- *Neural-Network based Methods*

In this research, we experiment with all three types of methods. We start with node2vec as an underlying method for the algorithm and then move forward with testing SDNE and GGSNN.

3.2 Node2Vec

Node2Vec is a random-walk-based algorithm that aims to produce embeddings of each vertex of the graph by generating probabilistic random walks[27]. This method has two major phases.

3.2.1 Phase 1: Random Walk Generation

In this phase, based on the nodes and the weights of the edges between the nodes, probabilities of the next step are generated. Intuitively, this means that in each unit of time, the algorithm moves one step forward in the probabilistic direction. It quantifies the relationship with each next step that it would take along with the distance of it from the starting point.

The probability function is as follows:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{z}, & \text{if } (v, x) \in E \\ 0, & \text{otherwise} \end{cases} \tag{3.2.1}$$

Here, π_{vx} is the transition probability between the nodes of v and x. This probability is then normalized by z. This is only true if there is an edge between the nodes. If there is no edge, then the probability goes down to zero. This process happens iteratively. We start with a node where the probabilities of all the neighbors are calculated and the neighboring node with the most probability is then selected; this process happens continuously till we reach the desired length of the random walk.

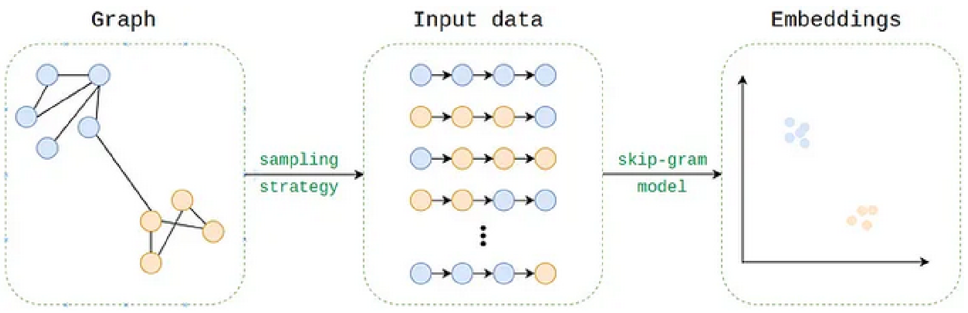

Fig. 3.2.1: Architecture of Node2Vec[27].

3.2.2 Phase 2: Embedding Generation

After a number of random walks have been generated. These walks are then treated as text. I.e. each node in the sequence is treated like a word and each sequence of a walk is treated as a sentence. These sentences are then subjected to the underlying embedding method, Word2Vec[52]. In detail, the Skip-Gram model is used for training the embeddings of the graph where the data from the random walks acts as the training data. Thus, in this case, the graph is represented as text. On the other hand, our research focuses on representing text in the form of graphs.

The very first random-walk-based methods was the Deep-Walk [57] method. It generated the walks without taking the probabilities of each step between two nodes. Thus, it was a truly random algorithm. There have been many improvements in this domain like NBNE [58] which samples direct neighbors of the subject node. Metapath2vec [15] is a follow-up method over node2vec that uses meta paths to generate the paths. While the path generation methods differ in each algorithm, their underlying neural network remains the same i.e. the Skip-Gram Model. However, there are a few methods like HSNL [17] and RMNL[80] that use Recurrent Neural Networks (RNNs) as the underlying method to generate embeddings.

For this research, Node2vec was chosen as one of the underlying methods because it provides simplicity in terms of time and space complexity while getting good results at the same time.

3.3 Structural Deep Network Embedding (SDNE)

SDNE[28] falls under the second type of graph embedding models, i.e. the algorithms that use Matrix Factorization to produce embeddings. It is one of the algorithms that preserve both, first-order and second-order proximity. Because of this, the algorithm is capable of preserving embeddings in a local and a global view. Unlike walk-based methods, it does not have multiple phases but the loss function of the neural network model in this algorithm incorporates multiple inputs. SDNE is a semi-supervised graph embedding architecture with two phases, each focusing on first-order proximity and second-order proximity, respectively. Thus, two different loss functions add up to produce a comprehensive loss function.

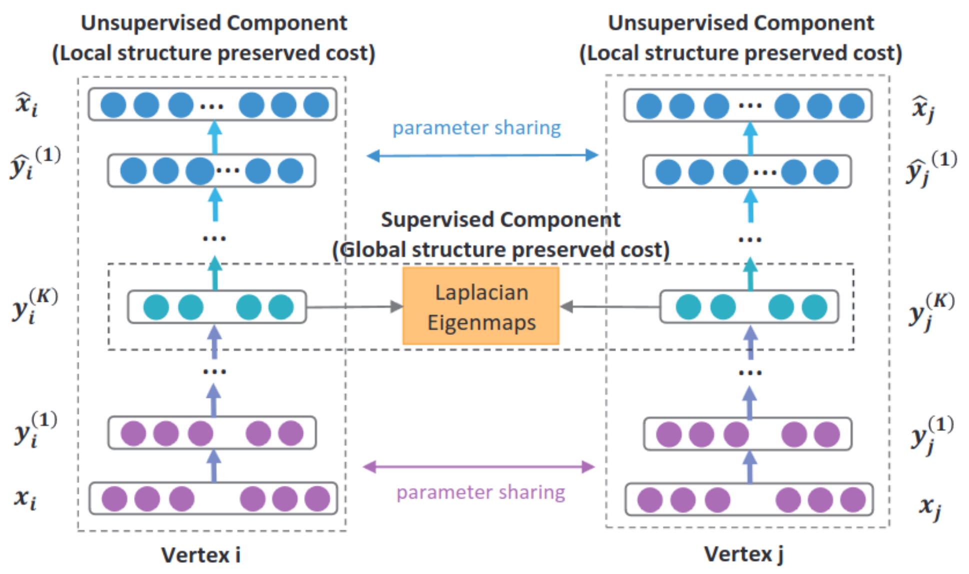

Fig. 3.3.1: The architecture of SDNE[28].

3.3.1 First-order proximity using Laplacian Eigenmaps

To incorporate the global structure of the graph, the entire adjacency matrix of the graph has to be worked upon. For this, the authors of SDNE introduced a supervised component that utilized the entire structure of the graph. The first phase of SDNE uses Laplacian-Eigenmaps to preserve the global node similarities. Laplacian-Eigenmaps impart a higher penalty if two similar nodes are embedded farther apart. So, the first loss function is defined as follows:

$$L_{1st} = \sum_{i,j=1}^{n} (s_{i,j} |y_i - y_j|^2) \tag{3.3.1}$$

Here, $s_{i,j}$ is the adjacency matrix and y_i is the subject node and y_j are the neighbor nodes of y_i.

3.3.2 Second-order proximity using Auto-Encoders

For the second phase, the SDNE[71] architecture utilizes an autoencoder[62]. It is the unsupervised component of the algorithm. An autoencoder contains two neural networks that work in synchronization with each other. The input of the encoder is an adjacency matrix of the graph which is mapped to a representation space with fewer dimensions and then the decoder takes in the encoded representation and reconstructs it to a reconstruction space. During the reconstruction, the nodes with a similar neighborhood will have similar embedding. Thus, the local proximity (First-order proximity) is preserved in this architecture. The loss function of the autoencoder is as follows:

$$L_{2nd} = \sum_{i=1}^{n} |(\hat{x_i} - x_i) \odot b_i|^2 \tag{3.3.2}$$

Finally, the SDNE combines the two above-described loss functions and the resultant loss function is a semi-supervised algorithm that needs to be minimized. The resultant loss function is as follows:

$$L_{mix} = L_{1st} + \alpha L_{2nd} + \mu L_{reg} \tag{3.3.3}$$

Where, L_{reg} is just a regularization parameter to prevent overfitting, μ is a parameter to control the regularization and α is used to control the influence of the first loss function on the overall loss function.

Unlike [31, 33, 77] which only make use of Matrix Factorization methods, this method proposes a method that works on both principles of supervised and unsupervised learning and that makes this method achieve better results than the ones prior to it at the expense of computational cost.

3.4 Gated Graph Sequence Neural Networks

This algorithm falls under the neural network-based methods for graph embedding. Before understanding GGSNNs, we need to understand the functioning of a Graph Neural Network (GNN)[25, 65].

In simple words, a GNN is a simple transformation that happens over the graph structure which optimizes the graph while preserving the attributes of the graph at the same time. Intuitively, a graph is taken as an input for the GNN where it is transformed progressively layer by layer with more and more information being learned (about the nodes, edges, and sub-graphs) each time. The resultant output is the embedding for each of the components of the graph, preserving all the connections of the graph.

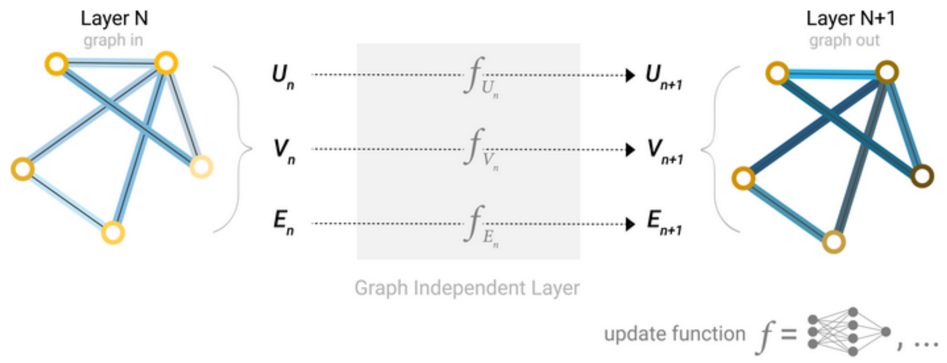

Fig. 3.4.1: The architecture of a simple GNN[25].

A simple GNN can be constructed by using a separate MLP for each of the components of the graph. Meaning that each node, edge, and global-context vector of the graph will have an MLP associated with it that generates its embeddings. Since all the components are treated separately in this method, the connectivity of the graph is not considered.

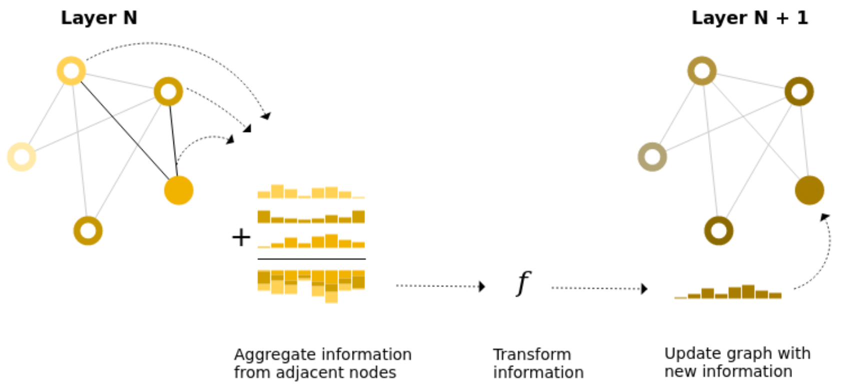
Layer N
Layer N + 1
+
f
Aggregate information from adjacent nodes
Transform information
Update graph with new information

Fig. 3.4.2: Message Passing in a GNN[24].

To incorporate the connectivity of the graph, a method known as **message passing**[24] is used on top of the simple GNN architecture. Message passing simply means to transfer information between the connected components of the graph (for instance: neighboring nodes and edges of a node).

The message-passing occurs by gathering information about all the neighboring components of a node and using an aggregate function on that information. This aggregate information is then updated using an update function which is usually learned through the learning process of the GNN.

For a task that deals with an input that is sequence-based and is in the form of a graph the Gated Graph Sequence Neural Network (GGSNN)[47] was introduced. Instead of a simple MLP, it uses a Gated Recurrent Unit (GRUs)[7] and its optimization techniques. As GRUs handle sequence-based output along with an attention mechanism, this algorithm has an inductive bias towards the problems that have sequence-based inputs.

Mathematically:

Say, a lot of messages are passed to node n_j. To quantify them along a simple MLP, we use the following equation.

$$F(x_j) = \sum_{j \in N_i} W_j x_j \tag{3.4.1}$$

To aggregate all the messages coming to the node n_i, suppose we use the function G.

$$agg_i = G(W_j x_j : j \in N_i) \tag{3.4.2}$$

After that, the node is updated by the GNN layer after gaining information about its neighbors. The update function is as follows:

$$h_i = \theta(K(H(x_i) + agg_i)) \tag{3.4.3}$$

Here, K and H are the MLP functions or GRUs, depending on the architecture. H is the initial transform that happens over the information gain, while K acts as a projection of vectors into another dimension.

Other neural network-based approaches include the Graph Convolutional Neural Networks (GCNs)[38] that function on pooling the information of the neighbors along with a Convolution layer to generate the embedding.

CHAPTER 4

Methodology

The task of generating vector embedding for words according to our method is a semi-supervised task because it involves a step where the graph needs to be created from a text corpus, which is essentially assigning relations to non-relational data. This solution involves 3 key tasks: (1) Graph creation, (2) Graph Enhancement and (3) Vertex Embedding.

4.1 Task 1: Graph Creation:

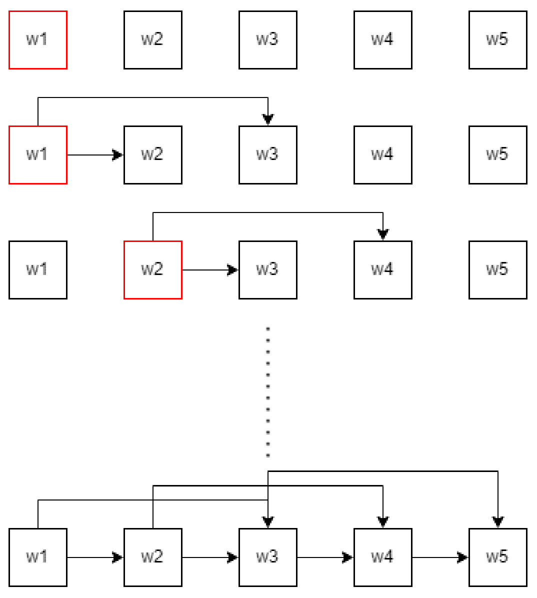

Fig. 4.1.1: A working example of the forward-pass algorithm on 5 words w_1 to w_5 with context size = 2

The objective of this method is to model the text corpus as a graph where every node corresponds to a word and the edges between the nodes signify the relations between the nodes. To convert the text corpus into a graph, we follow a simple **forward-pass** algorithm.

Figure 1 shows the nature of the forward-pass algorithm. In this example, there are 5 words (w_1 to w_5) that occur one after the other in the corpus. Here, we introduce a user-defined variable: **Context Size**. According to the context size, with every iteration, the algorithm will construct an edge between the subject word and the neighboring words in its context length. In the above example, the context size is 2, hence, it will associate the next two words of the corpus with the subject word. This is highly similar to how Word2Vec [52] captures context.

Whenever we add an edge to the adjacent matrix, its weight value depends on the decreasing function of $\frac{1}{X}$ as we go away in the context. Where X is the distance between the subject word and the neighbor word. The intuition behind this is that the closer the word is from the subject word, the stronger the association. This intuition for Context Size is similar to the Context Size in the Word2Vec algorithm [52].

Assumption: In our case, the distance between the subject word and the neighbor words always has to either zero or positive. As distance is a non-negative quantity, it cannot be less than zero. [6.2.1]

$$Penaltyfunction = \frac{1}{x}, x \geq 0 \tag{4.1.1}$$

This step takes care of maximizing the first-order proximity of the matrix. [6.2.1]

4.2 Task 2: Graph Enhancement:

The Adjacency Matrix for the graph we get from Task 1 is a spare matrix with not many associations between words. This is true for most word embedding models as well. So, in this task, our goal is to enhance this adjacency matrix so that the output is a denser, much more comprehensive adjacency matrix. To achieve this goal, we try to maximize the second-order proximity. [6.2.1]

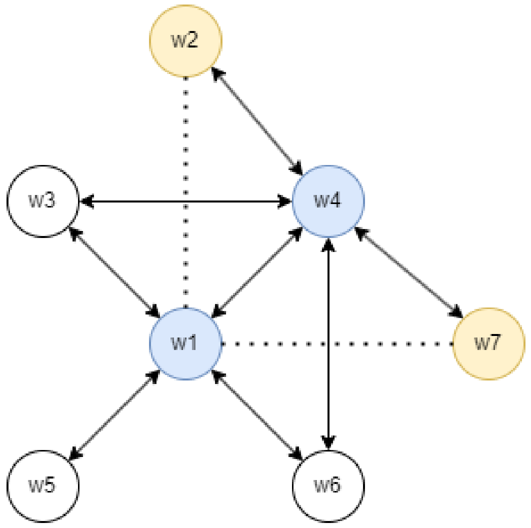

Fig. 4.2.1: An illustration that shows the co-occurrence between w1 and w4 is high. Thus, we create invisible edges between (w1,w2).

For this purpose, we compare the neighbors of the subject node and the neighboring node. The idea is that if the neighbors of two nodes are highly similar (co-occurring), then the nodes might be similar and the subject node can have some implicit connection with the neighbors of the neighboring node. This intuition is in line with the Second Order Proximity of a Graph mentioned above. Thus, this algorithm enhances the second order proximity of the adjacency matrix.

In Figure 2, the subject node w_1 is the subject node, and w_4 is the neighbor node that is under consideration. Here, out of all three neighbors of the w_1, two are also the neighbors of w_4, meaning that 66.67% of the neighbor nodes of w_1 co-occur as the neighbor nodes of w_4. This concludes that other neighbors of w_4 might also have some association with the subject node w_1. We define co-occurrence as follows:

Definition 1 ***Co-occurrence*** *Co-occurrence between two nodes n_1 and n_2 can be defined here as the ratio of familiar neighbors of n_1 and n_2 to the first-order of nodes of the graph.*

$$Co-occurence = \frac{Neighbors_{n1} \cap Neighbors_{n2}}{Neighbors_{n1} \cup Neighbors_{n2}} \tag{4.2.1}$$

In the above paragraph, if an edge is added between the subject node and the

Algorithm 4.2.1 The Graph Enhancement Algorithm

```
Input: Adjacency Matrix ($Adj\_Matrix$)
Input: $THRESHOLD > 0$
  for $i$ in $range(0, length(Adj\_Matrix))$ do
      $Primary \leftarrow Adj\_Matrix[i]$
      for $j$ in $range(0, length(Primary)$ do
          $Secondary \leftarrow Adj\_Matrix[j]$
          $I = Primary \cap Secondary$
          $U = Primary \cup Secondary$
          $C = \frac{I}{U}$
          if $C > THRESHOLD$ then
              for $j$ in $range(0, length(Secondary)$ do
                  $Adj\_Matrix[i][k] \leftarrow Adj\_Matrix[i][k] + \frac{1}{Context\_Size+1}$
              end for
          end if
      end for
  end for
```

new node, the weight of the new edge is $\frac{1}{ContextSize+1}$. This is because this new edge is added with the understanding that there might be an association between the two words(or nodes). Thus, its weight has to be less than the edges that already exist.

4.3 Task 3: Node Embedding:

This step involves using an underlying method that uses the aforementioned node embedding algorithms. Up until this step, we successfully represent the textual data in a graphical format. This allows for the use of conventional embedding methods to generate vector representations of each node (i.e. each word). It is known that there are three categories of embedding methods that can be used[76]:

- Matrix Factorization Methods
- Walk-Based Methods
- Neural Network Based Models

We use the state-of-the-art method of each kind for the scope of this research.

4.3.1 Walk-Based Method: Node2Vec

As talked before3, Node2Vec[28] takes in a graph as an input, generates random-walks based on the probability function, and generates the embeddings using the underlying Word2Vec algorithm. This would mean that initially, the text data is converted to a graphical representation, and after generating the walks, the nodes are again used as words on which the embeddings are generated.

Due to the nature of the algorithm, its performance of it would potentially be comparable to just the Word2Vec method. As this algorithm generates walks based on the probability, the immediate neighbors are almost always selected; thus the first-order proximity is preserved with Node2Vec. On the other hand, there is a possibility that not all the nodes in the global context vector get selected; thus the second-order proximity cannot be preserved in all cases.

4.3.2 Matrix-Factorization Method: Structural Deep Network Embedding (SDNE)

To win over the shortcomings of Node2Vec, Structural Deep Network Embedding (SDNE) [71] can be chosen. The SDNE architecture3 can be considered as a hybrid method that includes a matrix factorization method to preserve the global occurrence matrix (second-order proximity) and an auto-encoder architecture that preserves the connectivity of the local neighborhood (first-order proximity).

In our research, using this semi-supervised algorithm would mean that the positioning of the vectors of each word would be generated based on not only considering the context but also how that word acts with all other words in the vocabulary of the dataset. This would potentially result in gaining hidden connections between some words that do not exist in the context of the subject word.

4.3.3 Neural Network Based Model: GGSNN

A Gated Graph Sequence Neural Network[47] model is able to capture the semantics of the nodes from all the neighbors surrounding the subject node. Looking at it intuitively, each connected node transfers its information to every one of its neighbors resulting in every node in the graph having a global context of the graph along with a complete understanding of the neighborhood of the nodes.

Thus, this method preserves the first-order and second-order proximity of the graph. Additionally, because the GGSNN uses the GRU as its computation unit instead of a simple MLP from a GNN, this is best suited for sequence-based input. In our case, since the textual data is connected and it is proven that using LSTM[79] units or GRU[7] units on data that is sequence-based (i.e. texts), better embeddings can be generated [47], it would be best suited for our case.

Since this method does not involve training of node vectors in multiple steps, while preserving the same first-order and second-order proximity of the graph, this method can serve the same purpose as SDNE while potentially being better in terms of computational cost and time complexity.

CHAPTER 5

Experiments and Results

5.1 Experimental Setup

To evaluate our method, we subjected all the algorithms to the following intrinsic and extrinsic evaluation tasks. The training was done on the 20 News Groups Dataset[42], containing over 18,000 news articles from 20 different topics. Because the dataset is very diverse in terms of news coverage, the vocabulary is a lot varied.

The details of the test bench are as follows:

- CPU: Ryzen 7 5700x, 8 cores 16 threads
- GPU: NVIDIA Geforce RTX 3070, 8GB VRAM, 5888 CUDA Cores (CUDA enabled)
- RAM: Corsair Vengeance 32 GB @ 3600 MHZ, CL18 Timing
- Motherboard: ASUS Prime Gaming Motherboard, AMD B550 chipset
- Storage: WD Black PCIe Gen 4 SSD, 1 TB

To evaluate the algorithm, we construct a case study involving a lot of tests to evaluate the embedding method with different angles. The intrinsic tasks and the extrinsic tasks are as follows:

5.1.1 Intrinsic Tasks

- **Word Similarity:** This task involves investigating the relatedness of two words depending on the algorithm and comparing it with the human golden truth.

In this test, a similarity score using the cosine similarity 6.2.1 is calculated between two words. This pair of words would already exist in another dataset that has been assigned a similarity score created using human intuition. The similarity values between the two words would then be compared using a metric correlation function and that correlation would then give us an exact idea of how good our algorithm is. The datasets used as golden truths for this were the Simlex-999 dataset[30] and the WS353 dataset[1]. The Metric used for this test was Spearman Correlation.

- **Concept Categorization (Coref):** The task is to group similar words into similar categories. This test is analogous to a clustering problem [50]. In a conventional clustering problem, data points, based on their features are classified into groups. In our case, the data point is the word and its features are the vector embedding of that word. For example, *computer* and *phone* would belong to the same category. We evaluate this on the BLESS dataset[3] and the ESSLI dataset[2] as the golden truth datasets. The metric used for this test was Cluster purity.

- **Word Analogy:** The task is to determine the degree to which the semantic relations between word A and word B are similar to those between word C and word D, given the relations of A:B and C:D. For example, if we have to fill in the black in the sentence *'man is to woman as king is to ___'*; a human brain can probably guess queen for the blank, as it is analogous in this sentence. For an NLP model to recognize this analogy, the vectors must be generated in such a way that the words *queen* and *woman* are close and opposite to *king* and *man* respectively. This is the task of word analogy. We use the SemEval-2012-Task2[37] dataset and the MSR dataset[53] on the metric of Spearman Correlation.

5.1.2 Extrinsic Tasks

- **Question Answering(QA):** The task in this test is to select an answer given a question. The answer would be chosen as a segment from the given passage in some cases, while in other datasets the answer can be a boolean (yes or no). QA has become increasingly popular in recent times with the advancement in IR systems and chat-based voice assistants[20]. In this test, a model is trained based on the embeddings generated by the selected algorithm. The input of the model would be a question and the output would be the answer. In an extraction-based QA test like this one, the output would be the starting word index and the ending word index of the answer. The dataset used for this test was the SQUAD datset[61] and the model that was used for training was the Retrieval-Augmented Generation (RAG) model[46].

- **Co-reference Resolution:** In normal English texts, we often use linguistic expressions in place of the actual name of the entity to reference it. For example *I love for **Ironman** because **he** is fearless.* In this statement, the expression *he* refers to *Ironman* entity. For higher-level NLP tasks, this is a very important task. It involves grouping all the words that refer to the same entity in the corpus. We used the CONLL-2012 shared task[60] as the dataset, this dataset was trained on the model proposed by AllenNLP[22] with using embeddings generated by all the baselines.

- **Part-of-speech(POS) Tagging:** Any sentence in any language is made up of parts of speech (like verbs, nouns, etc.). In linguistics, these parts of speech of a word signify its definition and context. It is not as simple as making a list of words that correspond to a part of speech as there are many words that might be used in different parts of speech. Along with the understanding of word similarity from the embeddings, any word embedding model should also have good cognition about the context to get accurate POS tagging results. This task deals with tagging each word in a sentence with its syntactic role. We use the Penn-treebank dataset for this task[51] which is trained on the model proposed in [44]

- **Named Entity Recognition (NER):** It is the task of identifying the mentions of entities in the dataset into different categories like people, country, organization, etc. The NER is a very important step in data preprocessing for any NLP application. Most NER applications work based on a pre-existing graph database like DBPedia where the information is stored in triples just like a Knowledge Graph (KG). The CoNLL-2003[64] dataset was selected for this test. It has more than 250,000 tokens amassing over 25000 entities in the dataset. This was trained on the same [44] model.

5.1.3 Metrics

We chose the following metrics to evaluate the algorithms on,

- **Spearman Rank Correlation:** When the need arises to measure the similarity between two entities that are ranked, the Spearman Rank Correlation[74] can be used. It is an extension of the Pearson Correlation [9] metric. For two entities that are either ordinal, interval, or ratio and the relationship between them is **monotonic** 6.2.1, the Spearman correlation will be able to identify the strength and direction between the two entities/variables. Mathematically:

$$\theta = \frac{6 \sum d_i^2}{n(n^2+1)} \tag{5.1}$$

  Intuition: Suppose we have 2 lists of variables that need to be compared. The elements in the list would first need to be ranked (they are usually in ascending or descending order), and then the two lists of variables can then be compared based on the Spearman Correlation. In the formula, d_i is the difference between the values of the element that is ranked at i^th position. And n is the sample size of the dataset. In our case, this metric would be useful in conducting Word Similarity and Word Analogy tasks. In both of these cases, there would be a golden truth data that already has similarity values and analogy values between two words. These values are then compared to the similarity/analogy values that are predicted by the embedding algorithms. Based on these results, we would obtain the average values of Spearman Correlation for the tests.

- **Cluster Purity:** In a clustering problem 6.2.1, the metric of purity is a metric that is used for external evaluation of the clusters that are formed by the algorithm. It works on the principle of how many pure clusters are formed (i.e. clusters having only one class of data points). It is also one of the simpler evaluation metrics for clustering tasks. Mathematically:

$$purity = \frac{1}{N}] \sum max(m \cap d) \tag{5.2}$$

Intuition: The problem of clustering is an unsupervised learning problem. In this case, there is no target variable that an algorithm has to predict, instead it has to group all the similar kinds of data points into a cluster together based on their features. In this case, the simplest measurement would be to compare all the categorized data points with the ground truth and see how many data points have been categorized correctly. Suppose we have N data points that have d clusters and d classes, then with the intersection between all the classes and clusters, we would get all the data points that have been classified correctly. Normalizing this by N gives us the proportion of data points that have been clustered correctly. In our case, we use this metric for Concept Categorization. This problem is very similar to clustering as we are grouping similar words into clusters of a particular category. While the methodology differs from conventional clustering algorithms [50], the evaluation strategy is the same. After the words have been categorized by all the methods, they all would then be compared with the golden truth datasets which would give us the Cluster Purity score.

5.1.4 Baselines

We selected the following baselines to evaluate GraphWords:

- **Tf-iDF [35]**: This is the only statistical model in the selected baselines. It generates embeddings based on how relevant is a word compared to the document it is in. It is calculated by multiplying the frequency of a word in the document with the inverse document frequency of the word across multiple documents.
- **Word2Vec [52]**: We use the CBOW approach of the model where the embeddings are generated based on a task of a neural network. The task is to predict the subject word given its subject words in a context window. After the desired output is achieved, the model is frozen and the weights of the layers are the embeddings of the model.
- **GloVe [56]**: This approach makes use of a global co-occurrence matrix of words to generate embeddings. The co-occurrence matrix contains the probabilities of different pairs of words based on which the embeddings are generated.
- **SynGCN [69]**: It is a multi-phased word embedding approach that utilizes knowledge graphs. It uses the dependency parsing method to generate connectivity between the words. And then, pre-trained word embeddings are used and passed through a GCN to enhance the embeddings.

5.1.5 Models to be tested

Based on the underlying embedding methods, we select the following the models that are to be evaluated for the scope of the research:

- GraphWords + Node2Vec
- GraphWords + SDNE
- GraphWords + GGSNN

5.2 Results

This research aims to answer the following Research Question(RQ):

RQ: Can a graph-infused word embedding method incorporate not only the relatedness of words but also surpass conventional methods in that aspect (given similar semantic space)?

	Word Similarity		Concept Cat.		Word Analogy	
	Simlex-999	WS353	BLESS	ESSLI	SemEval	MSR
Word2Vec	21.0	43.5	64.3	53.2	14.2	32.4
TF-iDF	13.7	24.3	48.7	39.7	8.2	21.8
GloVe	31.1	4.8	73.9	51.4	14.0	35.9
SynGCN	34.9	41.2	79.2	**60.9**	18.6	41.2
GW+Node2Vec	37.2	45.7	78.5	59.4	19.3	43.9
GW+SDNE	38.9	47.1	**82.2**	59.4	18.1	44.6
GW+GGSNN	**39.0**	**48.3**	81.0	60.2	**19.9**	**44.9**

Table 5.2.1: Comparison of the Intrinsic tests on GraphWords with other baselines (Spearmann Correlation Coefficient for Word Similarity and Cluster Purity for Concept Categorization)

5.2.1 Evaluation of Intrinsic Tests

Table 5.2.1 shows the results obtained after conducting the intrinsic tests on the proposed algorithms and the baselines after training on the 20 Newsgroups dataset[42]. The results show that on the Word Similarity task, our proposed graph representation method along with the GGSNN[47] architecture gets the best result over the previous best baseline (SynGCN[69]) by an average of 4%. Apart from that, the results

	Question Answering	Coref	POS Tagging	NER
Word2Vec	73.5±0.3	60.1±0.1	93.0±0.1	59.3±0.1
Tf-Idf	61.2±0.1	52.3±0.2	89.3±0.1	47.1±.2
GloVe	74.2±0.1	62.7±0.1	94.6±0.2	59.8±.1
SynGCN	74.6±0.2	63.4±0.3	**95.4±0.1**	61.7±.1
GW+Node2Vec	75.2±0.1	64.1±0.1	93.8±0.2	59.6±.3
GW+SDNE	76.7±0.1	65.0±0.1	94.2.3±0.2	60.4±.3
GW+GGSNN	**77.1±0.1**	**65.4±0.1**	95.0±0.2	**62.6±.3**

Table 5.2.2: Comparison of F1 Scores/accuracies6.2.1 of the Extrinsic tests on Graph-Words with other baselines

obtained by our method with SDNE[28] are also on par with SynGCN. This shows that neural network-based models and matrix factorization methods prove better results. The better results for this task can be attributed to the fact that representing words based on their occurrences in a graph format would result in exploring some relationships that conventional algorithms would not be able to.

For the task of Concept Categorization, our representation method with SDNE as an underlying algorithm performs better (by 4.5%) than the previous baseline on the BLESS dataset. While on the ESSLI dataset, the results are comparable (within the margin of error) to the best-performing baseline. An explanation for a comparable performance can be given by the nature of the dataset that was used for training and the nature of golden truths in the ESSLI dataset. This dataset has only a limited number of words that are divided into 9 categories, out of which a lot of them do not occur in the training dataset. Due to this scarcity of data, the cluster purity score suffers for this test.

For the Word Analogy task, our algorithm outperforms previous baselines on both SemEval[37] and MSR[53] datasets by an average of 4.6%. This proves that representing text as graphs also embeds similar words together while also preserving other relationships between the words.

5.2.2 Evaluation of Extrinsic Tests

The extrinsic tests show real-life application-based tests of all the algorithms. Table 5.2.2 tells us how our models perform compared to the baselines in all the extrinsic tests performed.

For the QA task, the proposed methods with all the underlying methods perform better as compared to the baselines. Comparing this to the intrinsic tests, it can be found that the graph representation method along with any of the selected node embedding methods proves to be better by more than 4%.

Co-reference Resolution is a task that is useful for distinguishing the references to the entities. For this test, a method has to rely on how well the embeddings recognize the entities and how close the speech(nouns, adjectives, etc.) tags are to that entity. With an improvement of more than 2.5% over the baselines with SDNE[28], the proposed method outperforms other methods.

In the POS tagging task, the results are based on how better the model is able to recognize which word is used and what kind of part of speech. As SynGCN architecture uses dependency parsing as a mode to generate edges between nodes, its performance is better than all the models in consideration. Even then, the proposed method along with GGSNN performs comparably to it.

The GraphWords architecture also provides a minor improvement (of about 1.5%) departing from SynGCN for the Named Entity Recognition task. But this is only true for the architecture with GGSNN as the underlying method.

Overall, if we compare both methods, we see that representing text as a graph and then using node embedding methods provides a valuable increase from other conventional methods. However, when all the proposed methods are scrutinized, it is found that using recent methods that involve graph neural networks (in this case, GGSNN) gives much better results as opposed to matrix factorization (SDNE) methods and walk-based methods (Node2Vec).

5.2.3 Statistical Significance Testing on the Results

To support the hypothesis that using graphs for Word Embedding Methods can be potentially better, we conduct a hypothesis test. For this, a default Hypothesis is defined as:

H_0 = No better results are obtained by GraphWords as compared to SynGCN[69].

The alternative hypothesis on this was taken to be:

H_a = Results obtained by GraphWords are efficient and more as compared to SynGCN[69].

The significance level (α) was selected to be 0.3.

Thus, to test the above-defined hypotheses, we take the p-value to be:

p(Undertaking Graph Enhancement Process — No better results are obtained)

$$p = \frac{tests\_on\_which\_graphwords\_fail}{total\_tests\_conducted} \tag{5.3}$$

By these calculations, the p-value comes out to be 0.2. Thus, we can ignore the default hypothesis as p-value $< \alpha$. And hence, we can say that the alternate hypothesis is true and the obtained results are statistically significant.

5.2.4 Time and Space Complexity Analysis of the Proposed Methods

In this section, we discuss the time efficiency of the proposed methods and how many resources they use on a system. To test the efficiency of the methods, we measure the time to train the embeddings using the methods in consideration. Keeping the CPU and the RAM size and speed the same for all the tests, we measure the time and the usage of RAM while the models are being trained.

	Time to Train (minutes)	RAM Usage (GB)
Word2Vec	11.5	1.7
Tf-Idf	7.4	0.9
GloVe	(pre-trained)	(pre-trained)
SynGCN	18.7	4.3
GW+Node2Vec	22.2	13.1
GW+SDNE	43.6	22.2
GW+GGSNN	39.0	16.4

Table 5.2.3: Time-Space Analysis of GraphWords compared with other baselines

5.2.4.1 Limitations:

From Table 5.2.3, we see that the most efficient method in terms of time-to-train is the Tf-iDF method. While it takes the least time to train, it also performs the lowest as well. On the other hand, the proposed methods take the highest time to train. Among the graph representation architectures, the method with Node2Vec takes the least amount of time to train, while the method with SDNE takes the most amount of time to train. While other conventional models like GloVe and SynGCN produce better results with time taken more than indexing methods but less than the proposed method.

On the aspect of Resource/RAM Usage, the GW method with SDNE architecture takes the most amount of resources. This can be attributed to the matrix factorization method followed by an auto-encoder architecture. Both of these methods take up more space and time to execute, hence, they take the most amount of time. On the other hand, the index method (Tf-iDF) accomplishes the task in the least amount of time but at the same time, it achieves the worst results.

In general, we see a trend of all the GraphWords methods taking a long time. The reason behind this is that in the Graph Enhancement Algorithm (Algorithm 4.2.1), there are two nested loops. Thus, making the complexity of the initial stage of the algorithm O(n^3). As the number of words increases in the corpus, the algorithm would take more and more time. The overall high space usage of this algorithm can be explained by the fact that we need to store each word as a node in the graph and along with it, the edge information between the nodes as well. Since the order of edges increases exponentially as the number of nodes in the graph increase, it takes up a high amount of space.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

The tests of Word Similarity, Concept Categorization, Word Analogy, Question Answering, and Co-reference Resolution are all dependent on how well the algorithm captures the relatedness between the words. While the POS tagging task also takes into consideration some grammatical characteristics. This observation leads to the conclusion that our algorithm is able to capture context and relatedness between words in a better manner as compared to other methods.

So, to answer the RQ, we can say that a Graph-based word embedding algorithm can incorporate relatedness between words given sufficient context and can also surpass a lot of conventional embedding methods according to the conducted intrinsic and extrinsic tests. One advantage of using Graphs is that the vectors and words can be visualized as nodes in the graph, which can help towards the explainability of the methods.

But, the representation of words as nodes has some downsides as well that handicap the efficiency of the proposed method. As graphical databases either use an adjacency matrix or other storage-heavy methods, the GraphWords method takes significantly more resources to store.

Further, as the graph enhancement method in its current state is un-optimized, resulting in a high time complexity of the algorithm, the proposed method is also very inefficient in terms of time as well.

Even then, the tests conducted on the proposed method tell us that representing words as nodes in a graph paves way for better context capturing and similarity as graph-based methods explore neighborhoods in a much more efficient way.

Apart from this, this also leads way for a stackable word vectoring framework as multiple trained word-vector graphs can then be stacked on top of one another by using link propagation algorithms to improve pre-trained global vectors, much like GloVe[56].

6.2 Future Work

In the future, consequent research can be encouraged in order to improve the efficiency of the graph enhancement step of the algorithm. Methods such as dynamic programming can be implemented to counter multiple nested loops as they exist right now in a non-optimized state.

Alternatives for the storage of node and edge information can be used to improve the resource efficiency of the algorithm. Instead of an adjacency matrix, hashmaps can potentially be a quicker option. Further, for bigger datasets, graph databases such as Neo4j can also be helpful and quicker instead of the conventional methods.

For embeddings graphs and sub-graphs, there are a lot more methods that are energy-based (like TransE[5], TransR[48], TransH[73]). While these methods cannot embed a single node into vectors, they can convert the embedding of a group of nodes and the relationships between them to a vector. Such kinds of embeddings can be extremely useful in tasks such as coreference resolution and word similarity and other tasks where grouping multiple words in a singular category is of importance.

6.2.1 A Stackable Pre-trained Word Embedding Framework

The main advantage of using a graph database is that new nodes and new edges can be added to it at any time. Once a corpus has been used to train the embeddings on it, the embeddings can be stored as node information in the database.

After the creation of a graph database based on one corpus, suppose we create another database that is based on other corpora. There would be a lot of words that

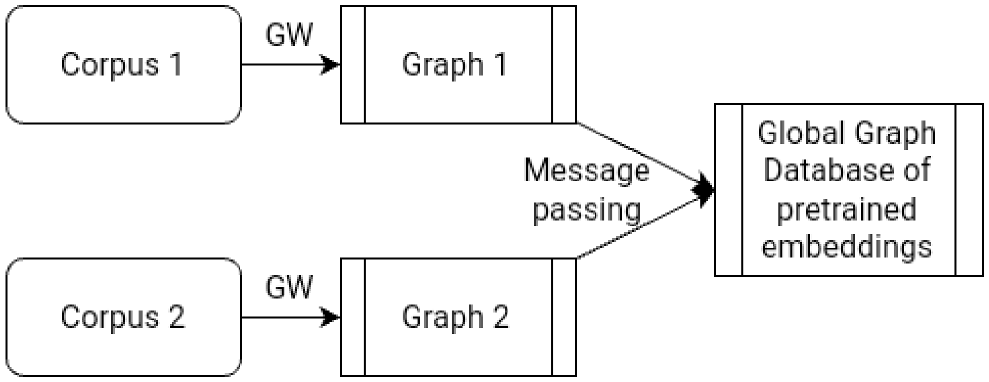

Fig. 6.2.1: Ideation of the stackable pre-trained word vector database based on Graph-Words

would be duplicated in both Corpora. These databases can then be easily merged into one graph by using the message passing architecture [23].

Thus, the more corpora we use to train the embeddings using the proposed methods, the bigger the pre-trained architecture of the words would get.

APPENDIX

First-order Proximity

The first-order proximity describes the pairwise proximity between vertexes. For any pair of vertexes, if $S_(i,j)$, there exists positive first-order proximity between V_i and V_j. Otherwise, the first-order proximity between V_i and V_j is 0. [71]

Second-order Proximity

The second-order proximity between a pair of vertexes describes the proximity of the pair's neighborhood structure.

Let $N_u = \{S_{u,1}, S_{u,2}, S_{u,3}...S_{u,v}\}$ denote the first-order proximity between V_u and other vertexes. Then, second-order proximity is determined by the similarity of N_u and N_v. [71]

Clustering

Clustering in Machine Learning means grouping unlabeled data points together. The clustering problem is an unsupervised problem as it does not have target variables pre-set. Instead, the data points are categorized based on their data points[50].

The $\frac{1}{x}$ function

The figure .0.2 shows the plot of the $\frac{1}{x}$ function. As the x variable increases, the corresponding y variable decreases to 0 progressively. Thus, we can say that this function is biased towards the values of x that are closer to 0 and imparts a penalty when the value of x increases.

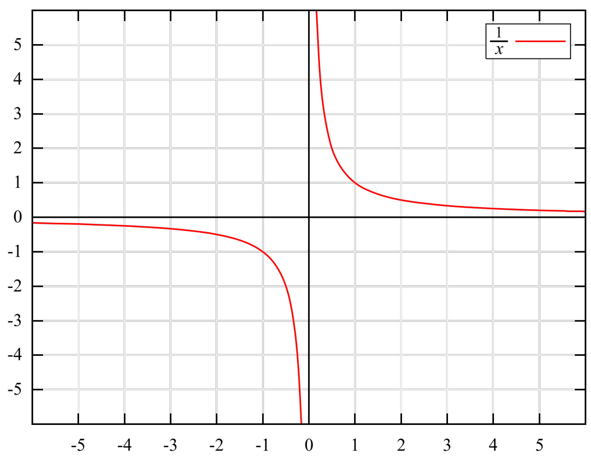

Fig. .0.2: The $\frac{1}{x}$ function.

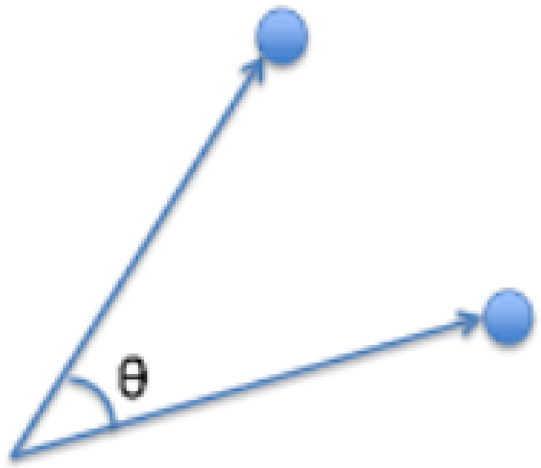

Fig. .0.3: Visual Representation of Cosine Similarity between two vectors.

Cosine Similarity

Cosine Similarity helps to tell how similar two vectors are. Suppose we have two vectors A and B in a given non-zero space. It is represented by θ. To measure the similarity between them, the mathematical formula would be:

$$\theta = \frac{A \cdot B}{||A||||B||} \tag{B.1}$$

Monotonic Functions

A function can be called monotonic if they are increasing or decreasing in their entire domain.

For example,

$$f(x) = x^2, x > 0 \tag{B.2}$$

$$g(x) = x^2, x < 0 \tag{B.3}$$

The function $f(x)$ can be considered monotonically increasing as it increases with the increase in the value of x. While the function $g(x)$ can be considered monotonically decreasing as it decreases with the decrease in the value of x.

Accuracy

Accuracy is a metric that is used for evaluating machine learning models on classification tasks. In a case where a model has to categorize data points in two classes, there are four possible cases; True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN).

Based on this, the accuracy can be defined as:

$$Acc = \frac{TP + TN}{TP + FP + FN + TN} \tag{B.4}$$

REFERENCES

[1] Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Pasca, M., and Soroa, A. (2009). A study on similarity and relatedness using distributional and wordnet-based approaches.

[2] Baroni, M., Evert, S., and Lenci, A. (2008). Lexical semantics: bridging the gap between semantic theory and computational simulation. In *Proceedings of the ESSLLI Workshop on Distributional Lexical Semantics*. Citeseer.

[3] Baroni, M. and Lenci, A. (2011). How we blessed distributional semantic evaluation. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, pages 1–10.

[4] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

[5] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

[6] Budanitsky, A. and Hirst, G. (2006). Evaluating wordnet-based measures of lexical semantic relatedness. *Computational linguistics*, 32(1):13–47.

[7] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[8] Clark, E. V. and MacWhinney, B. (1987). The principle of contrast: A constraint on language acquisition. *Mechanisms of language acquisition*, pages 1–33.

[9] Cohen, I., Huang, Y., Chen, J., Benesty, J., Benesty, J., Chen, J., Huang, Y., and Cohen, I. (2009). Pearson correlation coefficient. *Noise reduction in speech processing*, pages 1–4.

[10] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

[11] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537.

[12] Cui, P., Wang, X., Pei, J., and Zhu, W. (2018). A survey on network embedding. *IEEE transactions on knowledge and data engineering*, 31(5):833–852.

[13] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

[14] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[15] Dong, Y., Chawla, N. V., and Swami, A. (2017a). metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144.

[16] Dong, Y., Chawla, N. V., and Swami, A. (2017b). Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, page 135–144, New York, NY, USA. Association for Computing Machinery.

[17] Fang, H., Wu, F., Zhao, Z., Duan, X., Zhuang, Y., and Ester, M. (2016). Community-based question answering via heterogeneous social network learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[18] Fang, L., Luo, Y., Feng, K., Zhao, K., and Hu, A. (2019). Knowledge-enhanced ensemble learning for word embeddings. In *The World Wide Web Conference*, pages 427–437.

[19] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (2014). Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*.

[20] Floridi, L. and Chiriatti, M. (2020). Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694.

[21] Francis, W. N. and Kucera, H. (1979). Brown corpus manual. *Letters to the Editor*, 5(2):7.

[22] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M., Schmitz, M., and Zettlemoyer, L. (2018). Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.

[23] Gasteiger, J., Groß, J., and Günnemann, S. (2020). Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*.

[24] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

[25] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

[26] Goyal, P. and Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94.

[27] Grover, A. and Leskovec, J. (2016a). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

[28] Grover, A. and Leskovec, J. (2016b). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

[29] Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

[30] Hill, F., Reichart, R., and Korhonen, A. (2015). SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.

[31] Hofmann, T. and Buhmann, J. (1994). Multidimensional scaling and data clustering. *Advances in neural information processing systems*, 7.

[32] Islam, A. and Inkpen, D. (2008). Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):1–25.

[33] Jiang, R., Fu, W., Wen, L., Hao, S., and Hong, R. (2016). Dimensionality reduction on anchorgraph with an efficient locality preserving projection. *Neurocomputing*, 187:109–118.

[34] Jing, K. and Xu, J. (2019). A survey on neural network language models. *arXiv preprint arXiv:1906.03591*.

[35] Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

[36] Jurafsky, D. and Martin, J. H. (2014). Speech and language processing. vol. 3. *US: Prentice Hall*.

[37] Jurgens, D. A., Turney, P. D., Mohammad, S. M., and Holyoak, K. J. (2012). Semeval-2012 task 2: Measuring degrees of relational similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, SemEval '12, page 356–364, USA. Association for Computational Linguistics.

[38] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[39] Klimt, B. and Yang, Y. (2004). The enron corpus: A new dataset for email classification research. In *European conference on machine learning*, pages 217–226. Springer.

[40] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., and Brown, D. (2019). Text classification algorithms: A survey. *Information*, 10(4):150.

[41] Landauer, T. K., Foltz, P. W., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284.

[42] Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pages 331–339. Elsevier.

[43] Lebret, R. and Collobert, R. (2013). Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*.

[44] Lee, K., He, L., and Zettlemoyer, L. (2018). Higher-order coreference resolution with coarse-to-fine inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 687–692, New Orleans, Louisiana. Association for Computational Linguistics.

[45] Lewis, D. D., Yang, Y., Russell-Rose, T., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

[46] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

[47] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

[48] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

[49] Ling, W., Tsvetkov, Y., Amir, S., Fermandez, R., Dyer, C., Black, A. W., Trancoso, I., and Lin, C.-C. (2015). Not all contexts are created equal: Better word representations with variable attention. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1367–1372.

[50] Madhulatha, T. S. (2012). An overview on clustering methods. *arXiv preprint arXiv:1205.1117*.

[51] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

[52] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[53] Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

[54] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[55] Nadkarni, P. M., Ohno-Machado, L., and Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551.

[56] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[57] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

[58] Pimentel, T., Veloso, A., and Ziviani, N. (2017). Unsupervised and scalable algorithm for learning node representations.

[59] Pirrò, G. (2015). Explaining and suggesting relatedness in knowledge graphs. In *International semantic web conference*, pages 622–639. Springer.

[60] Pradhan, S., Moschitti, A., Xue, N., Uryupina, O., and Zhang, Y. (2012). Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint conference on EMNLP and CoNLL-shared task*, pages 1–40.

[61] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

[62] Salakhutdinov, R. and Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978.

[63] Salton, G. (1971). *The SMART retrieval system—experiments in automatic document processing*. Prentice-Hall, Inc.

[64] Sang, E. F. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

[65] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80.

[66] Sezerer, E. and Tekir, S. (2021). A survey on neural word embeddings. *arXiv preprint arXiv:2110.01804*.

[67] Su, C., Tong, J., Zhu, Y., Cui, P., and Wang, F. (2020). Network embedding in biomedical data science. *Briefings in bioinformatics*, 21(1):182–197.

[68] Tissier, J., Gravier, C., and Habrard, A. (2017). Dict2vec: Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263.

[69] Vashishth, S., Bhandari, M., Yadav, P., Rai, P., Bhattacharyya, C., and Talukdar, P. (2018). Incorporating syntactic and semantic information in word embeddings using graph convolutional networks. *arXiv preprint arXiv:1809.04283*.

[70] Verma, C. (2022). Chapter8 - corrosion and corrosion inhibition in neutral electrolytes. In Verma, C., editor, *Handbook of Science Engineering of Green Corrosion Inhibitors*, pages 69–78. Elsevier.

[71] Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234.

[72] Wang, Y., Hou, Y., Che, W., and Liu, T. (2020). From static to dynamic word representations: a survey. *International Journal of Machine Learning and Cybernetics*, 11:1611–1630.

[73] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28.

[74] Wissler, C. (1905). The spearman correlation formula. *Science*, 22(558):309–311.

[75] Wittgenstein, L. (2010). *Philosophical investigations.* John Wiley & Sons.

[76] Xu, M. (2021). Understanding graph embedding methods and their applications. *SIAM Review*, 63(4):825–853.

[77] Yang, Y., Nie, F., Xiang, S., Zhuang, Y., and Wang, W. (2010). Local and global regressive mapping for manifold learning with out-of-sample extrapolation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 649–654.

[78] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

[79] Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270.

[80] Zhao, Z., Yang, Q., Cai, D., He, X., and Zhuang, Y. (2016). Expert finding for community-based question answering via ranking metric network learning. In *Ijcai*, volume 16, pages 3000–3006.

VITA AUCTORIS

NAME:	Mrulay Mistry
PLACE OF BIRTH:	Ahmedabad, Gujarat, India
YEAR OF BIRTH:	2000
EDUCATION:	BE, Information and Communication Technology Science, Gujarat Technological University, 2021
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2023