Heidelberg University
Institute of Computer Science
Database Research Group

Master's Thesis

Learning Joint Vector Representations of Words and Named Entities

Name: Shideh (Satya) Almasian
Matriculation Number: 3335542
Supervisor: Prof. Dr. Michael Gertz
Date: October 30, 2018

Ich versichere, dass ich diese Master-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe und die Grundsätze und Empfehlungen “Verantwortung in der Wissenschaft” der Universität Heidelberg beachtet wurden.

_____

Abgabedatum: October 30, 2018

Zusammenfassung

Aufgrund der technologischen Fortschritte der letzten Jahre, verbunden mit einer enormen Menge an Textinhalten, die jede Minute veröffentlicht werden, hat sich die Erforschung der automatisierten Verarbeitung natürlicher Sprache zu einem wichtigen Bereich in der wissenschaftlichen Forschung entwickelt. Da ein Wort ein einzelnes bedeutungsvolles Element innerhalb einer Sprache oder eines Satzes sein kann, spielt die Erfassung seiner Semantik eine wichtige Rolle für das Verständnis von Sprache. Worteinbettungen (engl. word embeddings) sind eine Vektordarstellung von Wörtern, die Wörter eines hochdimensionalen Vokabulars auf Vektoren mit reellen Zahlen in einem niedrigdimensionalen Raum zuordnen, so dass Wörter mit ähnlicher Bedeutung auf beieinander liegende Punkte abgebildet werden, zum Beispiel gemessen anhand der euklidischen Distanz. Worteinbettungen werden als Features in vielen Informationsabruf- und Natural-Language-Processing (NLP) Aufgaben verwendet. Während viele solcher Aufgaben benannte Entitäten (engl. named entities) beinhalten, werden sie bei bekannten Worteinbettungsmodellen bisher nicht als eigenständiges Konzept erkannt und behandeln stattdessen alle Wörter gleichermaßen.

Obwohl es intuitiv scheint, dass die Anwendung von Worteinbettungstechniken auf ein mit benannten Entitäten versehenen Korpus zu intelligenteren Wortmerkmalen führen sollte, führt diese naive Herangehensweise zu einer verschlechterten Leistung im Vergleich zu Einbettungen, die auf rohen, unannotierten Text trainiert werden. Darüber hinaus erhöht das Kommentieren des Korpus die Komplexität der Beziehung zwischen Wörtern. Verschiedene benannte Entitätstypen haben unterschiedliche Auswirkungen auf die Wortsemantik, die normale Worteinbettungen nicht erfassen können. In dieser Arbeit stellen wir neue Ansätze vor, um gemeinsam Wort- und Entitäteneinbettungen in einem großen Korpus mit automatisch annotierten und verknüpften Entitäten zu trainieren. Darüber hinaus erweitern wir unsere Ansätze, um Beziehungen eines Wortes zu bestimmten Arten von anderen Entitäten in separaten Komponenten zu erfassen, was eine interpretierbare Darstellung erzeugt und die Bedeutung von Entitätstypen für eine Wortsemantik erläutert. Weiter diskutieren wir zwei Ansätze für das Einbetten von Trainingseinheiten, nämlich das Trainieren von modernen Worteinbettungstechniken für annotierten Text sowie das Einbetten der Knoten einer Co-Auftritt-Graphendarstellung (engl. co-occurrence graph) eines annotierten Korpuses. Wir nutzen die Vorteile dieser Graphenstruktur, um die Einbettungen mit trennbaren Komponenten vorzustellen, wobei jede Komponente die Beziehung des Wortes zu benannten Entitäten eines bestimmten Typs erfasst. Um diese separierbaren Komponenten zu erhalten, modifizieren wir gängige Einbettungen für Graphen und Wörter, um den Einbettungsraum in Typen der benannten Entitäten zu unterteilen, die im Text vorhanden sind.

Abschließend vergleichen wir unserere Ansätze mit denen klassischer Worteinbettungen, die mit dem Rohtext und einer Vielzahl von Wortähnlichkeits-, Analogie- und Clustering-Evaluierungsaufgaben trainiert wurden. Wir untersuchen außerdem die möglichen Vorteile und Anwendungsfälle solcher Modelle mit einer unternehmensspezifischen experimentellen Analyse.

Abstract

Because of the technological advances of recent years and vast amount of textual content being released every minute, natural language processing has grown into an exciting area of scientific research. With the goal of learning and understanding human language content has become more significant. Since a *"word"* is single distinct meaningful element of speech or sentence, capturing its semantics plays an important rule for understanding human language. Word embeddings are a vector representation of words, which map the words of a high-dimensional vocabulary to vectors of real numbers in a low-dimensional space, such that words with similar meaning are mapped to nearby points, measured in terms of the Euclidean distance, for example. Word embeddings are used as features in many information retrieval and natural language processing tasks. However, while many such tasks involve or even rely on named entities, popular word embedding models so far fail to recognize them as a distinct concept of their own and rather treat all words equally. Although it seems intuitive that applying word embedding techniques to a corpus annotated with named entities should result in more intelligent word features, this naive approach results in a degraded performance in comparison to embeddings trained on raw, unannotated text. Moreover, annotating the corpus increases the complexity of the relationship between words. Different named entity types have a different impact on the word semantics, which normal word embeddings fail to capture.

In this thesis, we propose novel approaches to jointly train word and entity embeddings on a large corpus with automatically annotated and linked entities. Furthermore, we extend our approaches to capture the relation of a word to a specific types of entities in separate components, which creates a more interpretable representation and shines some light on the significance of entity types for a word semantics. We discuss two approaches for training entity embeddings, namely training of state-of-the-art word embeddings techniques on annotated text, as well as embedding the nodes of a co-occurrence graph representation of an annotated corpus. We take advantage of this graph structure to introduce the embeddings with separable components, where each component captures the relation of the word to named entities of a specific type. To obtain these separable components, we modify well-established embeddings for graphs and words to divide the embedding space into types of the named entities present in the text.

Finally, we compare the performance of our approaches against classical word embeddings on a variety of word similarity, analogy, and clustering evaluation tasks. We furthermore explore the possible benefits and use-cases of such models with an entity-specific experimental analysis.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Dr. Michael Gertz for the continuous support, patience, motivation, and immense knowledge. His guidance helped me in all the time of writing of this thesis.
Furthermore, I owe my deepest gratitude to Andreas Spitz, without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. His insightful comments, encouragement, and instructions guided me in every step of the way.
I am also grateful to all the members of the IWR committee for allowing me the long and extensive use of their computing servers.
I would like to extend my thanks to Felix Feldmann, for his understanding and patience throughout these stressful times and for offering a helping hand in any difficult situation.
I am thankful to all the people who read this thesis completely or in parts. Great thanks to Dr. Bastian Rieck, for teaching me the proper ways of scientific writing and for always encouraging me to be better. I wish to thank my friends, Stefan Radev, Karsten Hanser and Gloria Feher for reading through this thesis even if the number of pages seemed daunting.
Most importantly, none of this could have happened without my family. I would like to thank my mother, my aunt, Ziba Mir-hosseini and Richard Tapper for providing me with the opportunity to follow my dreams and for their continuous support.

Thank you.

Contents

1 Introduction

A long-standing challenge for research in computer science is the understanding of written text and extraction of useful information from it. Text that is created by humans is often unstructured and ambiguous, whereas machine learning algorithms typically prefer fixed-length inputs and outputs. Thus, they cannot work directly with raw text directly and convert it into vectors of features.

Traditional *Natural Language Processing* (NLP) systems treat words as atomic units, represented as vectors, where each word is treated as a one-hot vector (with 1 in a single position and zero for the rest) to the size of the vocabulary. These vectors are sparse and orthogonal, and therefore, contain no notion of similarity among themselves. For example, if a user is searching for *"Heidelberg Hotel"*, documents containing *"Heidelberg Motel"* are to be disregarded since the dot product of the one-hot vectors of *"Hotel"* and *"Motel"* is zero. On the other hand, distributed vector space models represent words in a continuous vector space, where a word is represented by its relation to the surrounding words. These models are based on the idea that similar words tend to occur in a similar context or as the British linguist J.R. Firth said:

"You shall know a word by the company it keeps." (J.R. Firth 1957)

These distributed representations or so-called *wod embeddings*, map words of a vocabulary to a dense vector, such that words with closer meanings are mapped to the nearby points and the similarity between them is computed based on their distance in the embedding space. An example of the embedding space is illustrated in Figure 1.1, where the vectors of *"dog"*, *"pet"* and, *"labrador"* are mapped to nearby points and are farther away from unrelated words like *"snowboard"*.
Learning dense vector representations of words dates back to 2000 in the paper by Bengio et al.,

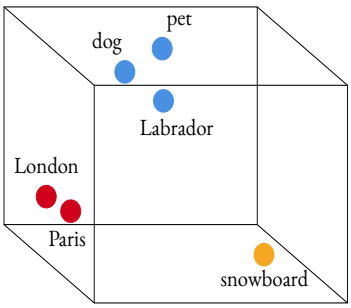


Figure 1.1: An example of dense vector representation of words in a 3-dimensional space, that maps words with similar meaning to nearby points.

where they presented a neural probabilistic language model [Bengio et al. 2000]. The first simple and scalable algorithm, however, is *word2vec* presented by Mikolov et al., where the aim is to capture the meaning of a word based on the surrounding context[Mikolov et al. 2013a]. There exist numerous variations of this model, from taking the character level information into account [Bojanowski et al. 2017] to incorporating the statistics of the whole corpus, namely the *GloVe* model [Pennington et al. 2014]. The GloVe model in particular, speeds up the learning and when trained on a large corpus tends to generate more promising results on downstream tasks such as named entity recognition.
Methods based on word2vec, despite being good at capturing semantics, have some drawbacks. They treat all words equally as terms and cannot be directly used to represent named entities. *Named Entities* (NEs) refer to proper names or nouns and numerical expressions, such as dates and quantities [Toral et al. 2008]. Many NLP tasks either depend on or use named entities as input. Identifying these references and classifying them has given rise to named entity recognition and disambiguation tasks. They are also extensively used in *Information Retrial* (IR) and search engines. Disregarding the named entities while generating a word embeddings creates several challenges for downstream tasks that use them as input. Named entities often contain compound words and phrases that if not grouped together from the start, result in ambiguous and unclear representations. Although some work has been done on embedding phrases [Hill et al. 2016; Yin and Schütze 2014] or combining compound words after training with averaging, adding, or multiplying [Durme et al. 2017; Mitchell and Lapata 2010], none of these focus on entities. The models that do focus on embedding entities, however, are designed for task specific applications, such as named entity disambiguation [Fang et al. 2016; Yamada et al. 2016]. These models are not only task-specific but also are focused on learning representations of *knowledge base* (databases containing information about named entities and their relation in a collection of documents) and do not focus on learning general purpose embedding of entities and words directly form a textual corpus.
A term-based model fails to distinguish between entity homographs such as *"Paris"*, which can refer to the French capital, the American heiress, or even the Trojan prince. Moreover, it cannot recognize compound word, as any word separated by space is considered its own token, whereas many companies and actors names consist of two word that not necessarily separate. In a term-based model, if an entity is repeated several times with different names, each name is treated independently that result in loss of valuable context information for the entity. Another drawback of term-based models is their inability to deal with date information. There exist multiple form of representing date and time in text, e.g, 1 January 1998 or 1998-01-01T15:53:00. Most models disregard any numeric information in text and limit themselves to a certain alphabet, with the exception of a few studies that try to add a temporal dimension to a paragraph or word embeddings, still they mostly focus on how the embeddings evolve or change with time and do not achieve a vector representation of dates [Mirza and Tonelli 2016; Yoon et al. 2018].
Another drawback of classical word embeddings in general is the lack of interpretability. In many cases, the semantic structure is heterogeneously distributed across the embedding dimensions, which makes interpretation challenging. Recently, some research has been done to make the embedding

less ambiguous [Faruqui et al. 2015; Park et al. 2017; Subramanian et al. 2018]. However, none of them show any insight about possible entity relations and most are applied as a post-processing step to the trained embedding. They offer no solution to learn the embeddings as an interpretable component during training time or an efficient method that can produce interpretable embeddings based on the entities surrounding a word. It is still unknown, what the implications are if a model predicts, for example, *"London"* and *"Berlin"* as similar. Dense vectors representing these words do not show the reason behind their similarity. It is not obvious, whether they both appeared in the same context location-wise, e.g., both are capitals and are situated in Europe or because they share the same organizations, e.g., many companies planned to move their offices from London to Berlin after the Brexit and are mentioned in the same context.
Since named entities play an important rule in many NLP and IR tasks, a more intelligent representation of them enhances the performance of systems that utilize them as input features. Word embeddings are used for named entity recognition, linkage, and entity ranking [Eshel et al. 2017; Lample et al. 2016; Ma et al. 2016; Siencnik 2015], where a unique and unambiguous representation of entities can boost their performance. Search engines, query expansion [Diaz et al. 2016; Kuzi et al. 2016], and document ranking [Zamani and Croft 2017] that utilize named entities can also benefit from a joint representation of words and entities. On the other hand, annotating the text with named entities, requires extensive pre-processing, from sentence splitting and tokenization to part-of-speech tagging. Not only are these steps prone to errors, but also the task of recognizing and classifying entities, even with the state-of-the-art tools, is far from perfect. Therefore, naively applying the word embedding methods to entity annotated text might cause more problems than benefits. It is important that the drawbacks of such an approach alongside its advantages are analysed.

1.1 Objective and Contributions

In this thesis, we address the problems of term-based models for generating embeddings for named entities as well as terms using an annotated corpus. In addition, to further analyse the importance of each entity type we experiment with faceted embeddings as a method to better understand the semantic structures and increase the interpretability of vector space models. Therefore, this thesis discusses two main models, one the entity and term embeddings and one for the faceted model. Below, we will discuss the contribution of each separately.
By analysing the entity-based models we make the following contributions:

- We apply the state-of-the-art word embedding methods on an annotated corpus to jointly train embeddings for terms and entities.
- We use graph embedding techniques to embed the nodes of a co-occurrence graph, extracted from an annotated text as an alternative means of obtaining entity embeddings.
- We evaluate both of our models against well-established word embedding methods on a set of intrinsic evaluation tasks.

- Since a few test datasets contain any entities, we create some of our own, to better analyse the effect of entity annotation.
- Through visualization and experimental analysis, we investigate the underlying semantics that entity embeddings capture, and discuss the implications for entity-centric downstream tasks.
- By comparing the entity-based and term-based methods, we identify their strengths, weaknesses and application scenarios.

To further analyse the impact of different types of entities on a word semantics and to make the embeddings more interpretable, we introduce faceted embeddings. With the faceted model, we make following contributions:

- We modify well-established word embedding techniques to include separable parts, where each part of the embedding corresponds to a specific types of entities.
- We use graph embedding techniques on a co-occurrence graph extracted from an annotated corpus to examine the relation of each word to the neighbours of a specific type. By embedding the nodes of such a graph we introduce an alternative way to achieve the faceted model.
- We evaluate our faceted models against the term-based methods on raw text, on a set of intrinsic evaluation tasks.
- Since separate parts of faceted embeddings divide the embedding space into the different type of entities available in text, we investigate the effect of each type on intrinsic tasks.
- By visualizing the embedding parts separately, we explore potential use cases and the underlying semantics of such models.

1.2 Overview of the Thesis Structure

The remainder of this thesis is structured as follows: In Chapter 2, we discuss the background information required for the model and related work, with a brief introduction to the basics of NLP and neural networks. This discussion is followed by, an explanation of well-established word embedding and graph embedding methods, which are the base of the entity-based models. In Chapter 3, we present two types of models for learning terms and entity embeddings jointly on an annotated corpus. One method focuses on learning the vector representation from a textual corpus, while the other method requires a co-occurrence graph extracted from the annotated text. Chapter 4 contains an in-depth definition of the faceted embeddings, where different models are proposed based on the input data (textual or graph-based). In Chapter 5, we present the test cases and evaluation results. The entity embeddings and faceted models are analysed using visualizations and also tested against well-known analogy, word similarity and categorization datasets. The work is closed with a conclusion and outlook on possible future work in Chapter 6.

2 Background and Related Work

To generate a vector representation of named entities and terms in an annotated document, we use either a corpus of text with entity annotations or the co-occurrence graph extracted from it. Unlike traditional methods, where all tokens are treated as terms, entity embeddings take the type information into account. To provide a general background for all the methods used in this thesis, the related work and background is organized as follows:
In Section 2.1 a brief overview of the text preprocessing steps for NLP tasks is given, in which methods used to extract information from text or common preprocessing steps are explained, including text cleaning, stemming, lemmatization, part-of-speech tagging, named entity recognition and linkage. In Section 2.2 neural network based approaches for learning embeddings are explained. Since word2vec and GloVe are the building blocks of other models, they are described in Section 2.4. In Section 2.5, we discuss the related work on generating entity embedding. In the same section, we also explain the generation of weighted adjacency matrix from a weighted co-occurrence graph, which is comparable to a co-occurrence matrix in the GloVe model. The co-occurrence graphs are explained in Section 2.3 along with the LOAD model [Spitz and Gertz 2016], which is a specific co-occurrence graph used in this thesis. Finally, an overview of the important graph embedding methods is given in Section 2.6, containing DeepWalk, node2vec, LINE, and VERSE. These models are used in Chapter 3 and 4 to create the entity and faceted embeddings, respectively.

2.1 Basics of Natural Language Processing

The goal of NLP systems is to understand and derive meaning from human language, with textual data as their main source of information. Most of the World Wide Web is made of text, with websites, such as Twitter generating vast amounts every day. Analysing this content is, however, not a simple task. For example, human generated text often contains ambiguity, as in the sentence *"I put my wallet in the car. It is green."* In this sentence, it is not obvious if *"it"* refers to the *"car"* or the *"wallet"*. Moreover, humans often use homographs (words that have the same spelling but different meanings), metaphors and sarcasm, which further increases the complexity of text analysis.
As machine learning systems typically rely on features, the most important step is to derive features from the text. Multiple models have been proposed for feature extraction, and while none of them achieve the goal of fully characterizing a text, their utility differs. However, all features require cleaning and preprocessing of the text.

2.1.1 Text cleaning

Before any operation can be performed on text, each sentence has to be broken down to its atomic pieces. *Tokenization* is the act of splitting sentences up into pieces, called *tokens*. For example, the sentence: *"Tokenization is widely used in natural language processing."*, can be tokenized as follows:

Tokenization	is	widely	used	in	natural language processing.

Splitting only by white spaces can also separate what could be regarded as a single token. For example, separating by white space would result into three tokens for *"natural language processing"*. In a later section, we discuss named entity recognition that tries to eliminate such problems.
Text data has also many inconsistencies that can cause troubles for algorithms. Some common initial preprocessing steps to decrease these inconsistencies are to convert all of the letters to lowercase and to remove punctuation. This makes sure that *"analytics"*, *"AnALYticS"*, *"Analytics!"*, and *"#analytics"* are all considered the same word. Removing punctuation should be applied with care, because in some cases, such as in the case of Twitter *"#analytics"* is a message about analytics and should not be confused with *"@analytics"*, which is a message to the analytics account. For these reasons, the removal of punctuation should be tailored to the specific problem.
Additionally, words such as, *"are"* and *"to"* are frequent in all documents but are only meaningful in a sentence. These are called *stop-words*. Despite the high frequency, these words are unlikely to result in a meaningful feature for a machine learning system and are often removed [Jurafsky and Martin 2000; Manning and Schütze 1999]. As with punctuation, removing stop words blindly may result in loss of valuable information. For example, *"The Who"* is the name of a band, and naive stop word removal might delete those words even if they carry significant meaning. Hence, removing all stop-words is not always helpful, but it is generally considered to be a useful step.

2.1.2 Stemming and lemmatization

Another important preprocessing step is *stemming* or *lemmatization*. This step is motivated by the desire to represent verbs with different grammatical conjugations as the same word. In many cases, there is no need for a distinction between *argue*, *argued*, *argues*, and *arguing*. They could all be represented by a common stem: *argu*. This is called stemming as it chops off the ends of words and often includes the removal of derivational affixes. Another approach is lemmatization, that uses the vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. Lemmatization is a more complex procedure as it requires detailed dictionaries, which the algorithm has to look through, to link a specific form back to its lemma. This is particularly problematic for irregular verbs, where the past and present tense of a verb do not share the same stem. For instance, the lemmatization of *"drank"* is *"drink"* [Jurafsky and Martin 2000; Larson 2010].

2.1.3 POS tagging: Part-of-speech tagging

POS tagging is the process of assiging a *part-of-speech* to each word in text. Parts-of-speech are also known as *word classes* or *lexical categories* are categories to which a word is assigned in accordance with its syntactic functions (e.g., noun, verb, adjective, …). POS tags are useful because they yield a large amount of information about a word and the syntactic context around a word [Jurafsky and Martin 2000; Manning and Schütze 1999]. Parts-of-speech are useful features for finding named entities, such as people or organizations in text [Jurafsky and Martin 2016]. Applications that provide POS tagging are also referred to as *POS taggers*, which are used to assign tags to tokens using a finite predefined tagset. For English, the most commonly used tagset is the Penn Treebank POS tagset [1].

2.1.4 Named entity recognition and entity linking

Named Entity Recognition (NER) or *entity extraction* is an information extraction method that locates and classifies the named entities in text. Named entities are words that can be classified into a set of pre-defined classes, such as the names of persons, organizations, locations, quantities and monetary values. In some cases, temporal expressions are classified separately as well. Generally, anything that is not an entity per se is considered to be a term [Nadeau and Sekine 2007]. More formally, if we define V as the set of all words in the vocabulary, $N \subseteq V$ is the set of all entities that fall into a pre-defined category. Recognition of entities is difficult, partly because of ambiguities and mostly because of multiple surface forms (different names for the same entity). It is not always clear what is or is not an entity or what type it has. For example, the entity *"Washington"* can be the name of a city, a person or even an organization. NER identifies the occurrence or mention of a named entity in the text, but it does not identify which specific entity it is. *Named entity linking* (NEL) or *named entity disambiguation* (NED) is the task of mapping the found entities to a repository of entities. For this purpose, *knowledge bases* or *knowledge graphs* are used, which contain rich information about the entities and their mutual relationships [Jurafsky and Martin 2000; Manning and Schütze 1999]. Knowledge graph is a repository of structured information consisting of unique entities, facts about entities ,and relations between entities. NEL maps the found entities to their corresponding entry in the knowledge base [Shen et al. 2015].

2.2 Neural Networks in Natural Language Processing

Before the advancements in deep learning and neural networks, most NLP techniques were based on machine learning approaches with linear models such as support vector machines or logistic regression, trained on very high-dimensional and sparse feature vectors. Recently, the field has changed to use non-linear neural-network models over dense inputs. The most important advantage of such methods is that they can often be trained with a single end-to-end model and do not require traditional task-specific feature engineering [Goldberg 2016]. Methods for different NLP tasks differ in

[1] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

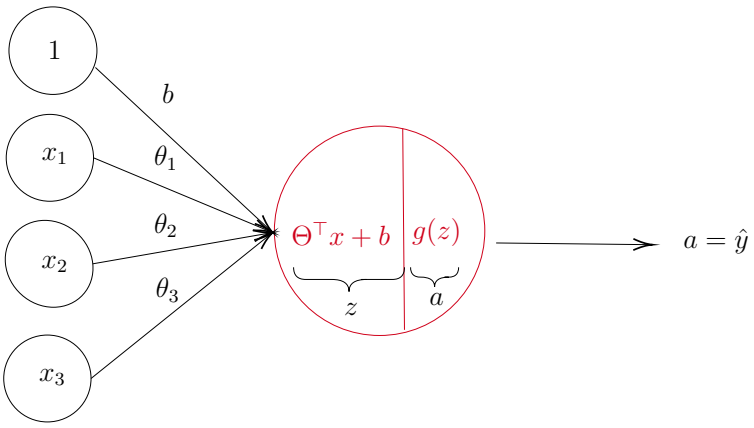


Figure 2.1: Single layer neural network. The input x is multiplied by the matrix of weights Θ to be passed to the activation function (typically a non-linear function to transform the input) to generate the output $\hat{y}$. [2]

architectures of neural networks and feature representations. Since the focus of this thesis is on word and entity embeddings, we focus on simple neural networks and the basic idea needed to generate word embeddings.

2.2.1 Neural networks

Despite neural networks being the new trend in machine learning, they are rather old algorithms, which are motivated by the goal of having a machine that mimics a brain. The sudden resurgence of neural networks is due to the fact that they are computationally expensive and require a huge amounts of data, which was neither available nor processable in the 80s and 90s. Also *"backpropagation"* algorithm [Rumelhart et al. 1986] and the advent of GPUs, sped up the training process and made it feasible to train models with thousands of parameters.

A single artificial neuron, consisting of a single computational unit, is shown in Figure 2.1. The computational unit illustrated in red takes in the different features of the input x_i and multiplies each with a given weight θ_i. The result of the multiplication is added to the *bias unit* b that always has an input value of 1. The weights indicate the importance of each feature for the computation and are learnable parameters of the model. The neuron then performs a transformation to the input through the *activation function* ($g(x)$) and generates the output ($\hat{y}$). The activation function is typically non-linear transformation that is applied on the input data.

Essentially, each neuron consists of two steps of computation. First, intermediate output z is computed by the multiplication of weights with input features, which is a linear operation that boils down to a matrix–vector product. Second, an activation function of choice is applied to z to produce the final output or activation (a) of the last unit.

A neural network is a group of multiple neurons connected together, as shown in Figure 2.2. In a neural network, the first layer is called the *input layer*, the last layer is called the *output layer* and all

[2] The figure is adapted from https://www.coursera.org/learn/machine-learning.

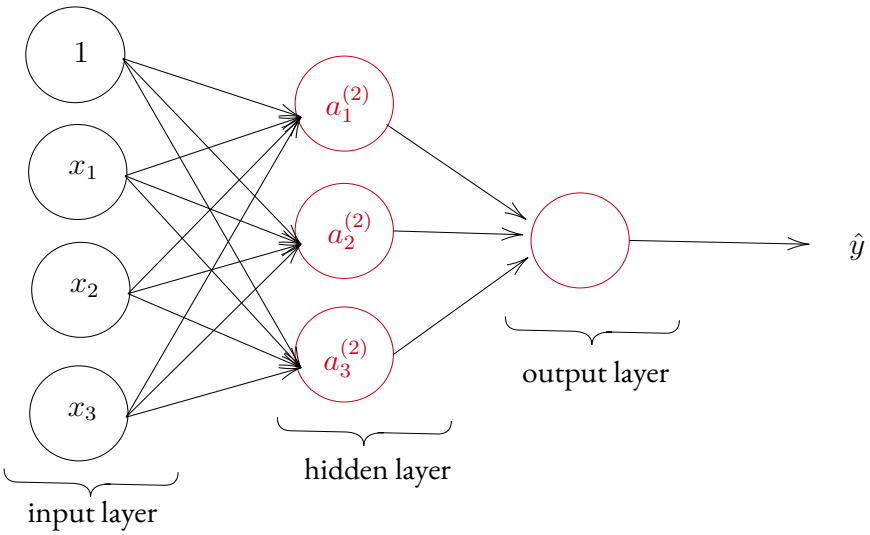
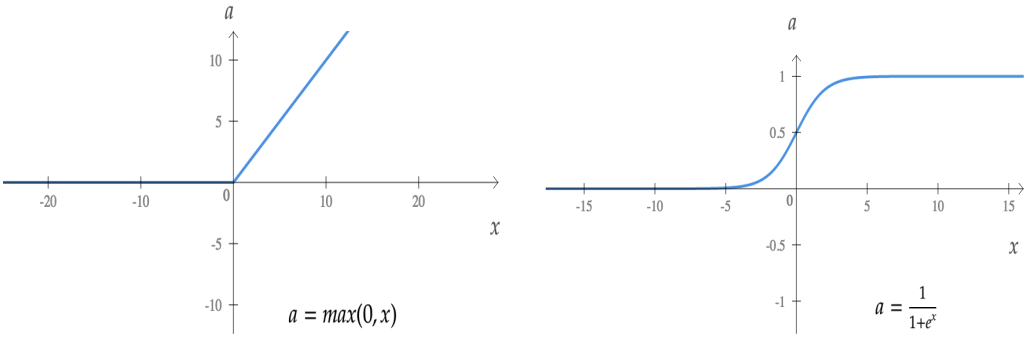


Figure 2.2: Two layer neural network with one hidden layer. An input x with three features is multiplied by the weight matrix Θ to generate an intermediate activation value a. The predicted output $\hat{y}$ is generated using the activations of the hidden layer. [3].

other layers are referred to as *hidden layers*. Typically, the values of the input and the output layer (labels) are given in the training set. The output of each computational unit in the network is the activation of that unit. We denote activation of the unit (neuron) i in layer j by $a_i^{[j]}$. The weights of the network are the parameters of the model and learning them is goal of any neural network. We denote the weight controlling the function mapping from unit n in layer j to unit l in layer $j+1$ by $\theta_{nl}^{(j)}$. The weights indicate how different features of inputs should be combined and how much should each of them influence the final output. The values of intermediate activations can be interpreted as latent features discovered during training [Haykin 2009]. As the network gets deeper, more complex combinations of input features can be learned.

An example of forward computation, which is called *forward propagation*, for the network in Figure 2.2 is given in Equation 2.1. Although a single output is shown the figure, the output layer can have different sizes. It can vary from one output for a single class classification even $10,000$ pixels of an image. The number of hidden layers can also vary, where the next hidden layer would use the activation of the pervious layer as input. How different neurons connect, the choice of activation function, the shape of the output layer and the number of layers is what defines different architectures [Goodfellow et al. 2016]. Since most of literature on word embeddings focus on shallow neural networks, in this thesis, we focus only on networks with a few hidden layers.

$$\begin{aligned} a_1^{[2]} &= g(\theta_{10}^{[1]}x_0 + \theta_{11}^{[1]}x_1 + \theta_{12}^{[1]}x_2 + \theta_{13}^{[1]}x_3) \\ a_2^{[2]} &= g(\theta_{20}^{[1]}x_0 + \theta_{21}^{[1]}x_1 + \theta_{22}^{[1]}x_2 + \theta_{23}^{[1]}x_3) \\ a_3^{[2]} &= g(\theta_{30}^{[1]}x_0 + \theta_{31}^{[1]}x_1 + \theta_{32}^{[1]}x_2 + \theta_{33}^{[1]}x_3) \\ \hat{y} = a_1^{[3]} &= g(\theta_{10}^{[2]}x_0 + \theta_{11}^{[2]}x_1 + \theta_{12}^{[2]}x_2 + \theta_{13}^{[2]}x_3) \end{aligned} \tag{2.1}$$



(a) (b)

Figure 2.3: Common choices of activation functions are (a) ReLU function $a = \max(0, x)$, which cuts values below zero and (b) Sigmoid function $a = \frac{1}{1+e^{x}}$, which forces the values to be between zero and one.

Activation functions are an important part of neural networks, without them the networks are just weighted sum of their inputs, plus a bias term that are unable to learn any complex and non-linear function. Activation functions are means to introduce non-linearity to the model. There are different classes of activation functions available but the most commonly used ones are *Sigmoid* and *ReLU*, both them are illustrated in Figure 2.3.

2.2.2 Training neural networks

Consider the output $\hat{y}$ as the prediction of our network for some task. In a supervised learning problem, the aim is to reduce the error between the prediction ($\hat{y}$) and the true label y. As a result, a cost function on parameters θ and b is defined as J in Equation 2.2, where L denotes the loss function on a single training example and m is the size of the training set. The *cost function* is simply the sum of all losses on the training set.

$$J(\theta, b) = \sum_{j=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \tag{2.2}$$

The loss for a single input is computed by the *loss function* L and the sum of loses for all inputs is computed by the cost function J. There exists various loss functions, which are chosen based on the problem at hand. We introduce three of the most commonly used loss functions, which are used by the models in this thesis:

1. *Mean Squared Error* (MSE) or *quadratic cost* minimizes the quadratic sum of distances between true label of each data point $y^{(i)}$ and its prediction $\hat{y}^{(i)}$ [Sammut and Webb 2017]:

$$L = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2 \tag{2.3}$$

[3]The figure is adapted from https://www.coursera.org/learn/neural-networks-deep-learning.

2. *Kullback Leibler (KL) Divergence* also known as relative entropy, is a measure of how one probability distribution $y^{(i)}$ (predicted distribution) diverges from a second probability distribution $\hat{y}^{(i)}$ (true distribution), the higher is the KL divergence, the more different the are distributions. KL divergence is not symmetric: The KL from $y^{(i)}$ to $\hat{y}^{(i)}$ is generally not the same as the KL from $\hat{y}^{(i)}$ to $y^{(i)}$ [Cover and Thomas 2006]. The formula consists of two parts, where the first part is called *entropy* and the second part *cross-entropy*.

$$\begin{aligned} L = \frac{1}{n}\sum_{i=1}^{n} D_{kl}(y^{(i)}||\hat{y}^{(i)}) = \\ \frac{1}{n}\sum_{i=1}^{n}(y^{(i)}\log(\frac{y^{(i)}}{\hat{y}^{(i)}})) = \\ \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}\log(y^{(i)}))}_{entropy} \underbrace{-\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}\log(\hat{y}^{(i)}))}_{cross-entropy} \end{aligned} \tag{2.4}$$

3. *Cross Entropy* is commonly-used where labels are assumed to take only two values and corresponds to the second part of KL divergence formula, where the second distribution $\hat{y}^{(i)}$ is fixed [Cover and Thomas 2006]. The cross entropy for a classification problem with two classes is defined as:

$$L = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})] \tag{2.5}$$

The goal of the training phase is to learn a weight matrix Θ and a bias term b such that the overall cost is minimized. This naturally leads to an optimisation problem, which is typically solved using *Gradient Descent* (GD) [Kiefer and Wolfowitz 1952]. Gradient descent is a way to minimize an objective function by finding local minima based on local gradients. It is an iterative algorithm, where in each step parameters of the model are updated using the gradient of the function. A single step of GD for one parameter θ can be seen in Equation 2.6, where the previous value of the parameter is updated based on the gradient of the cost function in respect with θ. Here, α is the *learning rate* or the step size and controls the magnitude of the update in each iteration.

$$\theta =: \theta - \alpha\frac{\partial J(\theta)}{\partial\theta} \tag{2.6}$$

In a dataset with m training examples, for each training example, the gradient of the cost function with respect to every weight and bias weight has to be calculated and accumulated. After iterating through all the training examples the weights and bias terms will be updated by the accumulate sum. This process is repeated for some number of iterations, where each iteration is one pass through the training set. Because the number of training examples for deep learning models are usually large, variations of GD have been introduce to speed up the process, three main types of GD are as follows:

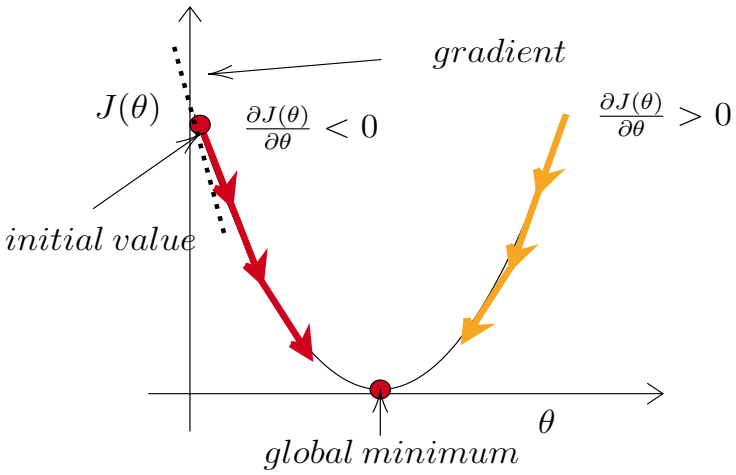


Figure 2.4: Gradient decent on a one-dimensional function with only one parameter. In each iteration, we take a step in the direction of the gradient. The gradient will guide the algorithm to the functions local minimum.

- Mini-batch GD: Instead of iterating over all training examples, a subset of a batch, i.e., a subset of the input data, is considered, before making an update, making it a good choice very large datasets.
- Stochastic-GD: In this case, only one example is looked at, before making an update.
- Batch-GD: The original algorithm with iteration over all examples.

Considering the most simple case (a function with only one parameter) gradient descent is illustrated in Figure 2.4. As shown in the figure, the derivative or the slope of the function is positive on the right side of the minimum (the update rule will decrease θ) and negative on the left size (the update rule will increase θ). The same rule holds in higher dimensions as well. Although the figure illustrates a function with global minimum, it is worth noting that GD is not guaranteed to find the global minimum and is often stuck in local minima.

2.2.3 Embeddings

One of the challenges of machine learning is to come up with suitable features for algorithms. These features often have to be task-specific and reflect the needs of the downstream tasks, for example, a task that uses embeddings as input features. *Representation learning* attempts to learn good features and representations automatically [Zhong et al. 2016]. *Principle Component Analysis* (PCA) [Shlens 2014] is one example of a traditional method for representation learning from high dimensional data. More recently, neural networks are used to obtain these low-dimensional representations. In a deep learning architecture, the output of each intermediate layer can be viewed as a representation of the original input data. Each unit inn the hidden layer of the network computes a non-linear combination of the inputs that is used by the next layer. If the dimensions of the hidden layer is smaller than

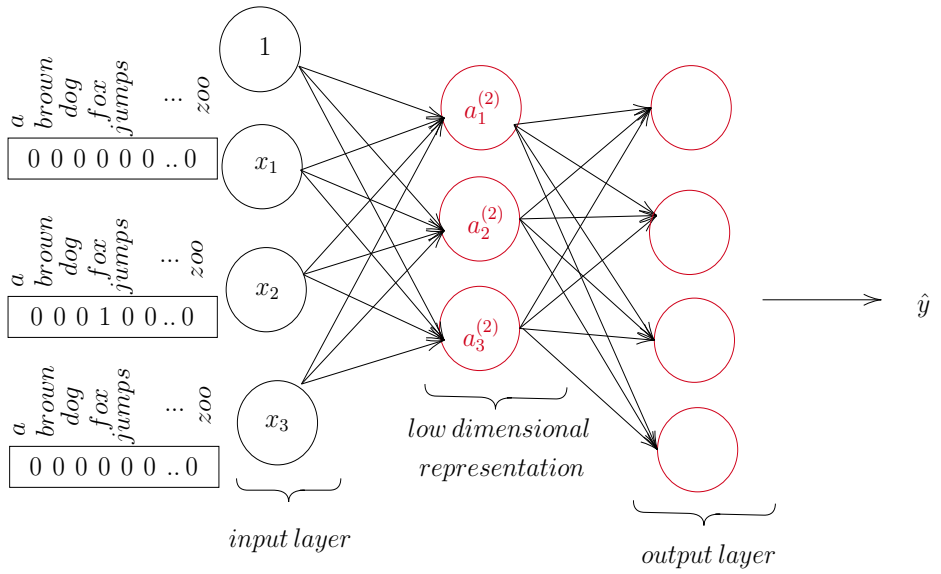


Figure 2.5: Underlying principle of word embeddings. The high-dimensional one-hot vectors are the input of the network. The hidden layer learns a dense representation to either predict the surrounding words or to optimise a different cost function in the output layer.

the input layer, a low-dimensional representation of the data is learned in the hidden layer. Based on this principle almost any type of data (e.g., images, sounds and textual content) can be transformed into embeddings with lower dimensions. In this work, we focus on learning low-dimensional representation for two types of data structures that are used by our models, namely, for words (*word embeddings*) and for graphs (*graph embeddings*).

Text is inherently high-dimensional, requiring a specific form of embedding. In most traditional NLP systems, each word is represented as a one-hot vector of the size of the vocabulary, which is a high-dimensional, highly sparse vector. By feeding these high-dimensional representations to a neural network with a smaller hidden layer, a new representation (usually in the order of 100 dimensions) for each word is learned, which is a more dense vector representation for each word. For this purpose, different models have been proposed, the most famous being word2vec [Mikolov et al. 2013a], which will be discussed in Section 2.4 in detail. All the models follow the same principle, which is illustrated in Figure 2.5, where the one-hot vectors of words in the vocabulary are the input of the network and a low-dimensional hidden layer is responsible for encoding the representation of input words. The size of the output layer depends on the particular architecture and the cost function. In addition, the network can be composed of multiple layers, but for our purposes, we focus on shallow networks with only one hidden layer.

Learning low-dimensional representations is not only a challenge for NLP, but extends to many domains, such as image and graph analysis. Graph embeddings, in particular, aim to convert a graph with many edges and nodes into a low dimensional latent space while preserving the graph information [Goyal and Ferrara 2018]. One type of graph embedding is *node embedding*, which represent each node of a graph as a vector in a low-dimensional space, where node proximity should be preserved in both spaces. The difference between various techniques lies in how they define *"closeness"* [Cai

et al. 2018]. For example, two nodes are considered close if they share the same neighbours or have a direct edge between them. In the next section, we will give a brief introduction to co-occurrence graphs, and in Section 2.6 the well-known graph embedding methods are explained.

2.3 Co-occurrence Graphs

Co-occurrence analysis, in terms of graphs, is not limited to text mining, but is a tool to find potential relationships between people, organizations, concepts, and biological organisms [Freilich et al. 2010]. A word co-occurrence graph shows the word interactions in a corpus. A co-occurrence graph $G = (V, E)$ is defined by a set of all words (V) as nodes and edges (E) between them, where there is an edge based on their paired presence in a unit of text [Nastase et al. 2015; Rousseau and Vazirgiannis 2013]. If the entities in the text are annotated, the graph shows the positional relationship between entities and terms that occur in the text. There is an edge between two words, if they occur in the same sentence or paragraph [Yin et al. 2018]. If the graph is weighted, the edges indicate the strength of the connection between two words or entities, which is related to the number of their co-occurrences or some kind of distance measure, e.g., number of tokens between them [Mihalcea and Radev 2011]. Since these graphs encode all important words in the vocabulary, they can form a graph-based representation of the corpus, which we use instead of a textual corpus to generate embeddings, where relations between nearby words can be extracted based on edges and nearby nodes.

The *LOAD model* [Spitz and Gertz 2016] is an entity co-occurrence graph representation of large document collections. For each document, named entities from sentences are extracted and connected based on their distance in a graph structure. The entity types considered by the LOAD graph are actors, locations, organisation, dates, and terms. The original LOAD model contains node types of pages and sentences as well, which we disregard for our models.
The LOAD model encodes the strength of relationship between entities and terms in the edge weights. In most cases entities on the same page share some connection, regardless of their distance in the text, which the models that only look for relation in a single sentence often miss. The LOAD model proposes a sentence-based weighting function for capturing the relation between two entities, where the long-distance connections have lower weights. Equation 2.7 shows the weighting of a single edge in the LOAD model between two entities v and s, where δ is a distance function. For entities other than terms, δ equals the number of sentences between the instances, or 0 if they occur in the same sentence. If the two instances of entities, namely i and j, do not occur on the same document their distance: $\delta(i, j) := \infty$. I_v and I_s are the set of all occurrences of v and s in the document. exp function forces the weight to diminish exponentially with the distance. Eventually, the sum of all these exponential distances creates the final weight.

$$e(v, s) := \sum_{i \in I_v j \in I_s} \exp(-\delta(i, j)) \tag{2.7}$$

The weights generated in this way encode the importance of one entity to another. The distance decays as the number of sentences between two entities grows. In addition, long-distance connections are considered very weak and are cut-off based on a threshold parameter. Despite the fact that related entities can be mentioned several sentences apart, terms are less likely to be related to entities outside of their own sentence, so the edges between terms and entities are limited to those that appear in the same sentence [Spitz and Gertz 2016].

2.4 Word Embeddings

A word embedding is defined as a mapping $V \rightarrow R^d : v \rightarrow w$ that maps a word v from a vocabulary V to a vector w in an embedding space of dimensionality d [Schnabel et al. 2015]. The first model to learn word embeddings as dense vectors is presented by Bengio et al., in which feature vectors, much smaller than the size of the vocabulary, are used to express words using a probabilistic model [Bengio et al. 2000]. The work of Collobert et al., in 2011, proved that the word representation could not only be obtained through probabilistic models, but that neural network architectures can also learn these internal representations from vast amounts of mostly unlabeled training data [Collobert et al. 2011]. However, it was not until the word2vec method [Mikolov et al. 2013a] that word embeddings became applicable to large corpora. Window-based models such as word2vec learn embeddings in terms of a supervised learning task, where the objective is to predict a word's context given a center word in a fixed window. A noticeable disadvantage of these models is that they do not operate directly on the co-occurrence statistics of the corpus. Instead, they scan context windows across sentences, which fails to take advantage of the vast amount of repetition in the data. In case of addition of new data, a window-based method has to parse the whole corpus again. On the other hand, a co-occurrence matrix can capture the previous co-occurrence in a compact way and hence, save time and computational power. Matrix factorization methods operate directly on the co-occurrence matrix and capture the full statistics. Before word2vec, similar embeddings were generated using *Singular Value Decomposition*(SVD) [Rohde et al. 2006] on co-occurrence matrices and keeping the top k dimensions. These methods are also able to capture many semantic and syntactic analogies. In 2014 Levy and Goldberg showed that implicitly factorizing a word-context matrix, whose cells are the *Point wise Mutual Information* (PMI) of the respective word and context pairs, can generate embeddings close to word2vec[Levy and Goldberg 2014c]. The main disadvantage of count based methods is that they are computationally slow on large matrices. Moreover, adding new words to the model is difficult, since it requires training a new model from the start.
The GloVe model [Pennington et al. 2014] combines the matrix factorization for generating embeddings with window-based methods and uses the global statistics in form of the word the co-occurrence matrix to generate embeddings. Furthermore, unlike the word2vec model that scales with the size of the corpus, general statistics of the data has to be generated only once in terms of the co-occurrence matrix and then additional computations can be performed on the matrix alone.

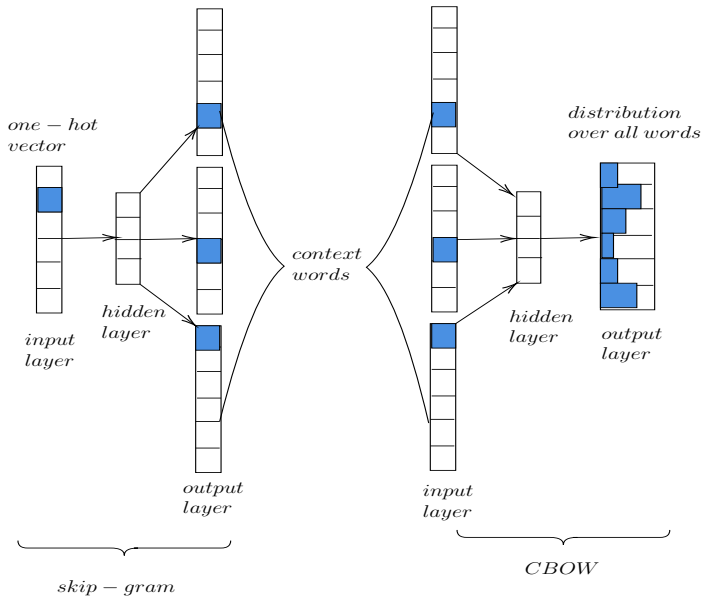


Figure 2.6: Predict a word, given the preceding and following words (Continuous Bag of Words, CBOW) and predict the preceding and following words, given a word (Skip-Gram). Image inspired by [Mikolov et al. 2013a].

2.4.1 Word2vec: Distributed representations of words

The goal of word2vec model is, given a center word, to predict the words that occur in its surroundings or vice versa. If sufficient data is used for training, word2vec can predict with high accuracy the word's meaning based on its surrounding words in the corpus and also successfully capture semantic relations, such as country and capital relations, as well as syntactic relations. For example, the vector representation of *"man"* has approximately the same distance to *"brother"* as *"women"* to *"sister"*. The word2vec method contains two models, the *continuous bag-of-words* (CBOW) and the *skip-gram* architecture. Both are shown in Figure 2.6. The CBOW architecture predicts the current word-based on the context and the skip-gram predicts surrounding words given the current word. While the two models are quite similar, they have different attributes. The CBOW smooths over a lot of the distributional information (by treating an entire context as one observation), which makes it useful for smaller datasets. However, skip-gram treats each context-target pair as a new observation and tends to do better on the larger datasets [Mandelbaum and Shalev 2016]. In the following, we will focus on the skip-gram model.

The skip-gram objective is to train word vector representations that are good at predicting the nearby words. A visual representation of the skip-gram model can be seen in Figure 2.7 for the sentence *"The quick brown fox jumps over the lazy dog."*, where the meaning of the word *"fox"* is defined by the words in the window around it. The input is the one-hot vector for the the target word and the output is the context words.

More formally, given a corpus of target words f and their contexts c, the goal is to learn the param-

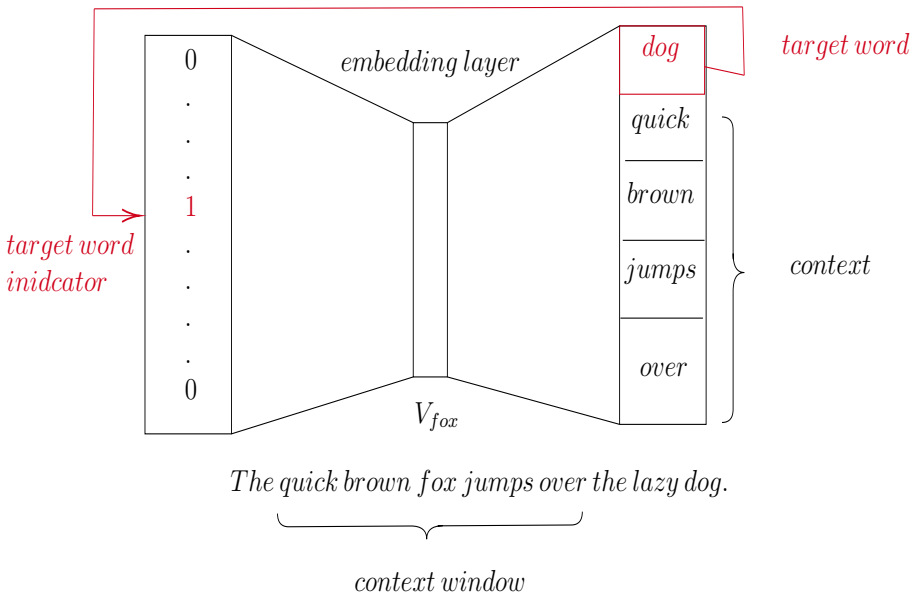


Figure 2.7: Skip-gram architecture for the example sentence *"The quick brown fox jumps over the lazy dog."*. The input is the one-hot vector indicating the word *"dog"* and the model tries to predict words that co-occur in the context window of the target word.

eter w of $p(c|f;w)$, so as to maximize the corpus probability, where Q is the set of all words and context pairs extracted from the text. Equation 2.8 shows the probability formulation.

$$w^* = \underset{w}{\text{argmax}} \prod_{(w,c)\in Q} p(c|f;w) \tag{2.8}$$

The literature on neural-network language models, defines this probability using softmax shown in Equation 2.9, where w_c and $w_f \in R^d$ are vector representations for c (context words) and f (focal or center word) respectively, and C is the set of all available contexts. The goal is to find the parameter w such that Equation 2.8 is maximized [Goldberg and Levy 2014].

$$p(c|f;w) = \frac{e^{w_c.w_f}}{\sum_{\overline{f}\in C} e^{w_c.w_{\overline{f}}}} \tag{2.9}$$

Maximizing the log-likelihood of Equation 2.9 on the training set is very expensive. Because we need to compute and normalize each probability using the score for all other words $\overline{f}$ in the context of c. Therefore, the authors reformulate the problem using *Negative Sampling* [Gutmann and Hyvärinen 2012]. Negative sampling is used in order to deal with the expensive computation of the softmax, in which the multinomial classification problem (predicting the next word) is converted into a binary classification problem. To estimate a true probability distribution of the output word, binary logistic regression is used, where the classifier learns to distinguish between a true pair (true context words) and randomly selected words from the vocabulary (corrupted pairs). The classifier simply predicts whether a pair of words is a true or a random sample. The new objective function, using negative sampling, is shown in Equation 2.10. σ shows the sigmoid function and $\overline{Q}$ is the set of random

Table 2.1: Co-occurrence probabilities for target words *"ice"* and *"steam"* with selected context words. Example taken from [Pennington et al. 2014].

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	large	small	large	small
$P(k\|steam)$	small	large	large	small
$\frac{P(ice\|k)}{P(steam\|k)}$	large	small	1	1

(f, c) pairs, assuming they are all incorrect. By maximizing this objective, the model assigns high probabilities to the real word pairs, and low probabilities to noise word pairs. Furthermore, it is more computationally appealing than the softmax function.

$$\underset{w}{\operatorname{argmax}} \sum_{(c,a)\in Q} \log \sigma(w_c.w_f) + \sum_{(c,\overline{f})\in \overline{Q}} \log \sigma(-w_c.w_{\overline{f}}) \tag{2.10}$$

If the model is trained on enough data, word2vec groups the vectors of similar words together in vector space, finds word's associations and detects similarities mathematically with cosine similarity.

2.4.2 GloVe: Global vector embeddings

The *Global Vector Model* (GloVe) suggests that the ratio of co-occurrence probabilities can encode meaning components. An example can be seen in Table 2.4.2, in terms of studying a thermodynamic phase. A meaning component that sets the word *"ice"* apart from *"steam"* is represented as a ratio of their occurrence probabilities with all other words $\left(\frac{P(ice|k)}{P(steam|k)}\right.$, where k can be any word in the vocabulary$\left.\right)$. This ratio can distinguish relevant words in comparison to irrelevant ones by giving the irrelevant ones a probability close to one. Also compared to the raw probabilities, it is better able to discriminate between the relevant words by giving the relevant ones a high ratio, such as words like *"solid"*, and low ratio to irrelevant ones, such as *"gas"*. As a result, the word *"ice"* stand out as a solid object. Moreover, words, such as *"water"* or *"fashion"*, therefore, has a probability close to one as it is either a shared property or not related to the concept.

To capture these ratios, the authors of GloVe propose a log-bilinear model, where, if the dot product of two vectors corresponds to the log of their co-occurrence probability, their difference will show the meaning component. The relation of the dot product of the word embeddings with the co-occurrence probability, for two word i and j, based on assumptions of the GloVe model is shown in Equation 2.1, where w_i is the word embedding for the word i.

$$\begin{aligned} w_i.w_j &= \log P(i|j) \\ w_x(w_i - w_j) &= \log \frac{P(x|j)}{P(x|j)} \end{aligned} \tag{2.11}$$

Let the matrix of word-word co-occurrence counts be denoted by $X \in R^{|V|\times|V|}$, where $|V|$ is the size of the vocabulary. Entries X_{fc} tabulate the number of times word c occurs in the context of word f. $w_f \in R^{1\times M}$ and $\tilde{w_c} \in R^{1\times M}$ are the focal (center word) and context embeddings, respectively, where M is the embedding size. The model tries to learn embeddings that minimize the squared difference between the dot product of the center and context word embedding and the logarithm of their co-occurrence count. Considering $\tilde{b_f}$ and b_c as the biases for the focal and context embeddings, the cost function is defined in Equation 2.12. Since the log of co-occurrences does not directly result in a probability, in order for the meaning component to have a probabilistic interpretation, a normalization factor is needed. Bias b_f allows the model to learn this constant during training and $\tilde{b_c}$ is added to preserve symmetry.

$$J = \sum_{f,c=1}^{|V|} f(X_{fc})(w_f^\top \tilde{w_c} + b_f + \tilde{b_c} - \log X_{fc})^2 \tag{2.12}$$

optimising for the co-occurrence counts alone might cause the model to overemphasize the most common words. Therefore, a weighting function (f) is introduced that imposes an upper bound on the maximum number of co-occurrences between two words. Conceptually, f scales the counts in order to avoid the influence of common words and boost the rare words. The choice of f is not fixed, but one class of functions that is used by the authors is parametrized with α as the exponential weight and x_{max} as the maximum number of allowed co-occurrences. α and x_{max} are hyper-parameters that require tuning, but the values suggested by the authors are $\frac{3}{4}$ for α is and 100 for x_{max}.

$$f = \begin{cases} (\frac{x}{x_{\max}})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{2.13}$$

The model learns two embeddings for each word, one as a focal or center word and the other as a context word. As the co-occurrence matrix is symmetric both of which can be considered as learning the same representation. Ultimately, the two embeddings are added to obtain the final embedding.

2.4.3 Similarity between embeddings

The dense word vector representation is learned to capture the semantics of a word, and hence, similar words are close in the induced space. *Cosine similarity* helps to capture this semantic closeness and is the most common similarity measure for word vectors. The cosine similarity is a measure that calculates the cosine of the angle between two vectors. This metric is a measurement of orientation and not magnitude and it is derived from the equation of a dot product between two vectors, as shown in Equation 2.14. The vectors are normalized by their length, which removes the influence of their magnitude on the similarity. The norm of the vector is somewhat related to the overall frequency of which words occur in the training corpus, but the direction is unaffected by this. For similarity between two words, it is often the direction that matters and not the overall frequency in the corpus. So in order for a common word like *"frog"* to still be similar to a less frequent word like

Table 2.2: LOAD weights for target words *"Donald Trump"* and *"Barack Obama"* with selected context words, where low values are denoted by blue and high values by red.

Probability and Ratio	$k = republican$	$k = democratic$	$k = president$	$k = BMW$
$P(k\|Trump)$	10.51	5.30	52	0
$P(k\|Obama)$	0.36	1.29	66	0
$\frac{P(Trump\|k)}{P(Obama\|k)}$	29.2	4.10	0.78	0

"Anura" (a type of frog), cosine similarity which only looks at the direction works better than simple Euclidean distance. Moreover, cosine similarity is symmetric and therefore, changing the order of vectors in the dot product does not affect the final result.

$$\begin{aligned} \overrightarrow{w_i}.\overrightarrow{w_j} &= \| \overrightarrow{w_i} \| \| \overrightarrow{w_j} \| cos\theta \\ cos\theta &= \frac{\overrightarrow{w_i}.\overrightarrow{w_j}}{\| \overrightarrow{w_i} \| \| \overrightarrow{w_j} \|} \end{aligned} \tag{2.14}$$

2.4.4 Edge weights and co-occurrence probabilities

The cost function of the GloVe model is based on the assumption that the ratio of co-occurrence probabilities results in meaningful components, which can be investigated to find the relation between two words. Edge weights of the LOAD model have a correlation with the co-occurrence counts of terms and entities in text. However, the sentence distance is a more efficient way to define a distance metric between two entities rather than a window-based approach. Two entities appearing in the same document tend to have some connection, this relationship is disregarded if we only look at a small window of words. Additionally, since the weights decay based on how far an entity is from another, we gain more information about the structure of the text. Based on these considerations, we argue that LOAD edge weights produce a better overall corpus statistics in case of named entities in comparison to a simple co-occurrence matrix and that it can generate the same meaningful components like probability ratios.

We demonstrate this claim with a simple example in Table 2.2 that shows how certain aspects of meaning can be extracted from edge weights of LOAD. Suppose we are interested in the concept of US presidents, for which we take two actors *"Donald Trump"* and *"Barack Obama"*. The relationship between these entities is examined by studying the ratio of their edge weights with various probe words. Similar to the GloVe model, where the ratio of co-occurrence probability distinguished the relevant words from irrelevant words, here the ratio of the edge weights works in the same way. In this case, looking at the raw edge weights does not give us much comparable information, but the ratio denotes *"Donald Trump"* as a republican rather than a democratic politician. Since both of the entities share the presidential attribute the ratio is close to one. On the other hand, a random name like *"BMW"* has a relatively small weight with both entities, but since the weak edges were cut off from the model we have the weight of zero. If the full graph without a cut-off threshold is

considered instead, the ratio of two small numbers will also converge to one, implying that similar to a co-occurrence matrix, edge weights of LOAD produce meaning components. As a result, the bilinearity assumption of GloVe can be extended to the weighted adjacency matrix of LOAD.

2.5 Entity Embeddings

Entity embedding are mostly studied in the context of embedding knowledge graphs in prepration of a downstream task, usually for entity disambiguation. Guo and Barbosa use random walks on knowledge graphs to construct vector representations of entities for named entity disambiguation task [Guo and Barbosa 2018]. He et al. use a deep neural network to represent the entities and their relations in knowledge graphs for inference tasks and link prediction in knowladge bases [Yang et al. 2014]. Aside from pure knowledge graph embedding, a few models focus on combining word embeddings with embeddings from knowledge graphs for enity disambiguation tasks. In a paper by Yamada et al., the skip-gram model trained on a textual corpus is combined with knowledge graph embeddings, where the vectors are aligned, such that similar words and entities occur close to one another in the vector space. The learned embeddings are then used as a feature for named entity disambiguation [Yamada et al. 2016]. Fang et al. also take a similar approach and by learning word embeddings from text and entity embeddings from a knowledge graph. They use an alignment model that guarantees the vectors of entities and words are in the same space and utilize them to design features for their disambiguation framework [Fang et al. 2016]. All of the above methods, although successful at representing entities and words are dependent on knowledge bases as an extra source of information and often optimise different objective functions for learning the entity and word embeddings, or train them separately and combine them in a later step. In this thesis, we avoid using separate training objectives for entities and word and treat them equally to learn embeddings directly from annotated text. Furthermore, learning embeddings for words and entities simultaneously, removes the necessity for an extra alignment model. It is also important to note that since the above methods are designed as feature for a specific task, they are only evaluated through the performance on that task. In contrast, we focus on creating general-purpose entity embeddings that are comparable to normal word embeddings. Additionally, we evaluate our embeddings against state-of-the-art word embedding techniques to asses their advantages and drawbacks in performance.

2.6 Graph-based Embeddings

Graph embeddings convert a graph into a low dimensional space in which the graph information is preserved. Therefore, it is an effective and efficient way to compute features from a graph for machine learning algorithms. Based on the embedding output, graph embeddings are categorized into four types: *node embedding*, *edge embedding*, *hybrid embedding* (combination of different types of graph components, e.g., node + edge), and *whole-graph embedding* (a graph is represented as one

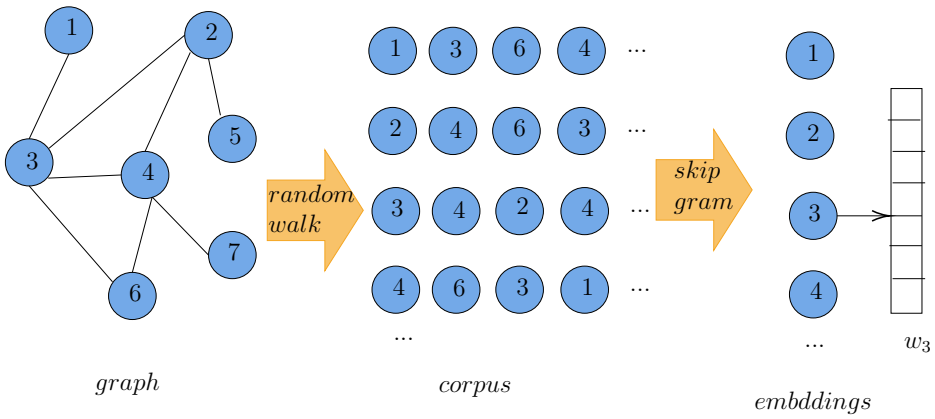


Figure 2.8: The workflow of DeepWalk. It first generates random walk sequences from a graph, and then applies the skip-gram model to learn the embeddings. Figure adapted from [Zhang et al. 2018].

vector) [Cai et al. 2018]. In this thesis, we focus on node embedding, as we use it to extract entity embedding form a co-occurrence graph in later chatpers. Node embedding techqniques embed nodes of a graph in a low-dimensional vector that is usually smaller than the number of total number of nodes in the graph. Different node embedding models vary in their definition of similarity between nodes and the graph property they preserve. Two most important properties are *first-order proximity* and *second-order proximity*. First-order proximity captures the direct neighbourhood relationship (edges) in a graph. Second-order proximity captures the second step relations (number of common neighbours shared between two nodes). The second-order proximity can also be extended to take higher orders into account as well (third-order proximity and so on) [Gutmann and Hyvärinen 2012]. Since the nodes of the co-occurrence graph correspond to words in the vocabulary, in this study we only look at node embedding techniques. Embedding the edges or whole graph does not provide any information about the words of the corpus. In the following we give a brief introduction into popular node embedding methods.

2.6.1 DeepWalk, a random walk-based embedding

The *DeepWalk* [Perozzi et al. 2014a] model learns latent representations of nodes of a graph using local information obtained from fixed length random walks. The model treats the nodes visited in the random walk as words and the walk itself as a sentence from the corpus. Then the skip-gram is applied on the generated corpus to maximize the probability of visiting the neighbourhood of the node conditioned on its embedding. A visualization of the workflow of DeepWalk is shown in Figure 2.8. In other words, the model tries to learn the embedding $w \in R^d$, where d is a small number of latent dimensions in comparison to the number of nodes. The model maximizes the probability of observing the last k nodes and the next k nodes in the random walk entered at node v_i, by minimizing the negative log of the probability as shown in the Equation 2.15, to learn the node embedding w [Goyal and Ferrara 2018].:

$$J = -\log P(v_{i-k}, ..., v_{i-1}, v_{i+1}, ..., v_{i+k}|w) \tag{2.15}$$

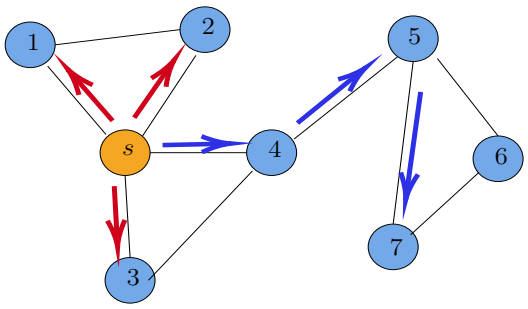


Figure 2.9: DFS and BFS sampling for a random walk with length 3 from the start node *"s"*. The figure is adapted from [Grover and Leskovec 2016].

Based on this approach, nodes that share similar neighbours in random walk sequences should be represented closely in the embedding space. Since the random walk in DeepWalk correspond to the *Depth-first Sampling* (DFS) , it can preserve the second-order proximity. As a result, DeepWalk is good at keeping community structures.

2.6.2 Node2vec: Scalable feature learning for graphs

Node2vec [Grover and Leskovec 2016] extends the random walk procedure of DeepWalk by defining a more flexible notion of a node's neighbourhood. There are mainly two node sampling strategies in random walk: *Breadth-first Sampling* (BFS) and *Depth-first Sampling* (DFS), where the BFS focuses on the immediate neighbours of the node and DFS consists of sampled nodes sequentially sampled at increasing distances from the source node [Grover and Leskovec 2016]. Therefore, BFS represents the first-order proximity and DFS the second and higher-order proximity. A demonstration of different sampling methods is provided in Figure 2.9, where red arrows illustrated the BFS startegy and blue arrows show the DF startegy for a random walk with lenght 3. Node2vec provides a trade-off between BFS and DFS sampling methods, with a use of two hyperparameters to regulate the balance between the them. By choosing an appropiate balance between the two sampling methods, we preserve community structure as well as structural equivalence between nodes [Goyal and Ferrara 2018]. Despite the flexiblity of node2vec, the method tends to be quite inefficient for large graphs and incurs significant space and time overhead. It runs out of memory even for mid-sized graphs, which is problematic for large co-occurrence graphs. Although there exist complex graph structures to speed up the process, the original model is very slow on large graphs [Zhou et al. 2018].

2.6.3 LINE: Large-scale information network embedding

The *Large-scale Information Network Embedding* (LINE) model [Tang et al. 2015] does not use random walks, instead it optimises two objective functions, one each for first- and second-order proximities, and minimizes the combination of the two to obtain two embeddings. The first objective function aims to keep the adjacency matrix and dot product of embeddings close. For the first objecitve function, the LINE model defines two joint probability distributions for each pair of nodes,

one using an adjacency matrix and the other using the embedding and minimizes the KL-divergence between two distributions, resulting in the first half of the embedding. The authors define another probability distributions and objective function for the second half of the embdding, which is based on the the second-order proximity [Goyal and Ferrara 2018]. Nevertheless, the model fails to learn meaningful representation of graphs with unbalanced edge weights, as the objective function for the second-order proximity is ill-defined when the edge weights have a high variance [Tang et al. 2015].

2.6.4 VERSE: Versatile graph embeddings from similarity measures

The *Versatile Graph Embeddings from Similarity Measures* (VERSE) [Tsitsulin et al. 2018b] model learns embeddings by training a single-layer neural network, which can be instantiated with diverse similarity measures. Given a graph $G = (V, E)$, where V is the set of all nodes and E the set of all edges, the aim is to learn the node embedding $w \in R^d$, where d is a small number of latent dimensions, by optimising the objective function in Equation 2.16. In optimisation objective, the KL-divergence from the given similarity distribution sim_G to that of sim_E in the embedded space is minimized, for any node v. The objective function is designed such that any similarity measure can be inserted in place of sim_G.

$$\sum_{v \in V} KL(sim_G(v, .), sim_E(v, .)) \tag{2.16}$$

Based on the Equation 2.16, the similarity of two nodes based the similarity measure G is related to their dot product in the embedding space. The similarity distribution (sim_E) between two nodes i and j, in the embedded space, is their dot product ($w_i.w_j$), normalized with softmax, as shown in Equation 2.17. To obtain the embeddings that satisfy an arbitrary similarity measure G on the graph, we should minimize the KL-divergence between the sim_E and sim_G for all nodes.

$$sim_E(v, .) = \frac{w_v.W^\top}{\sum_{i=1}^{n} \exp(w_v.w_i)} \tag{2.17}$$

Although VERSE is designed to accept any similarity measure, three measures are contained in the original implementation: *Personalized PageRank* (PPR), *Adjacency Similarity*, and *SimRank*. PPR is based on the stationary distribution of a random walk with restart. Thus, this measure is closely related to DeepWalk and node2vec. SimRank is a measure of structural relatedness. Based on SimRank, if two nodes are connected to similar nodes, they are themselves considered similar. Adjacency Similarity is based on the normalized adjacency matrix. More formally, given the out degree $Out(v_i)$ of node v_i, the Adjacency Similarity (sim_G^{ADJ}) is shown in Equation 2.18 [Tsitsulin et al. 2018b].

$$sim_G^{ADJ}(v_i, v_j) = \begin{cases} \frac{1}{Out(v_i)} & if \quad (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

Since the optimisation objective is expensive, the *VERSE* samples positive and negative samples with *Noise Contrastive Estimation* (NCE) [Gutmann and Hyvärinen 2012] to converge to a solution. NCE is closely related to Negative Sampling used in word2vec. The difference is that in word2vec words for the negative samples are drawn from a specially designed distribution, which favours less frequent words. In similar manner, instead of the expensive softmax computation, NCE trains a binary classifier to distinguish between node samples coming from the empirical similarity distribution sim_G and those generated by a noise distribution over the nodes.

In Chapters 3 and 4, we use the background information presented in this chapter about the word embedding and graph embedding techniques to define our entity embedding and faceted embedding models from entity annotated text.

3 Joint Entity and Word Embeddings

In this chapter, we explore different ways to jointly train word embeddings for entities and terms. We use the word and graph embedding on annotated data to generate such embeddings. In Section 3.1, an overview of all models is given and the objectives are explained. Section 3.2 is dedicated to models trained on unannotated (raw) text, that are used as the baseline against which we compare our models. The remaining two sections describe the two entity-based models proposed in this thesis. In Section 3.3, word embeddings that are trained on annotated text are explained, followed by graph-based models in Section 3.4.

3.1 Overview and Objectives

We introduce entity-based embeddings to address the issues that entity-centric downstream tasks would face, when using word embeddings as features. Any downstream task that needs a vector representation of entities has to either additionally perform entity recognition and linkage on normal word embeddings, or settles for using the ambiguous features that such models present. To address this problem, we learn embeddings for entities and terms directly from the text, without any post-processing steps. The pipeline for generation of entity-based embeddings is shown in Figure 3.1.
For word and graph-based models, we annotate the corpus in the first step and use it directly as input, while the graph-based models have an additional step of co-occurrence graph extraction. The nodes of the graph are entities and terms in the text; thus, as an alternative method we embed the

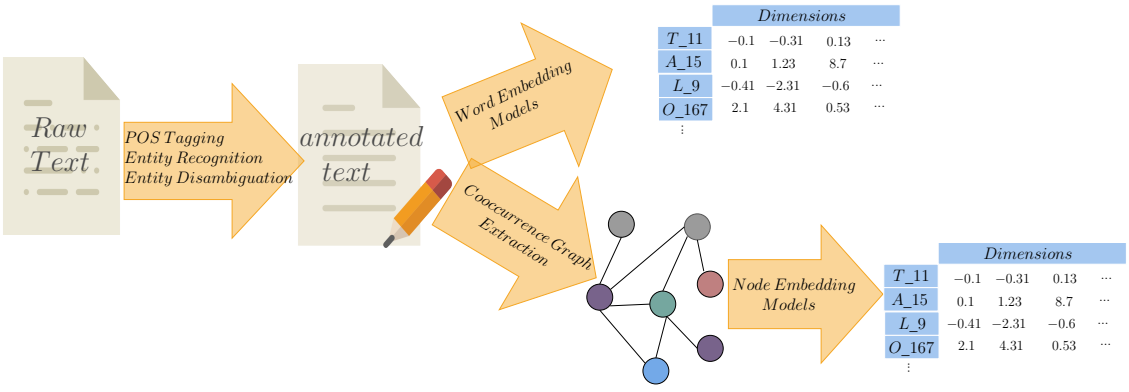


Figure 3.1: Pipeline for creating entity-based embeddings. Annotation of the raw text contains: POS tagging, entity recognition, and linkage. For textual models,Word embedding methods are applied to annotated text, while the nodes of a co-occurrence graph are embedded for the graph-based models.

nodes of the co-occurrence graph to obtain embeddings for all the words in the vocabulary. Generating embeddings from annotated text is advantageous for the following reasons:

- *Distinguishing homographs:* With annotation, we can distinguish between entity homographs, which written the same way but represent different entities. As a result, a vector representation of *"Washington"* as a person is different from the same word as a location.

- *Embedding compound words:* Unlike raw text, where the compound name is broken down into specific words, names of entities are grouped into a unique identifier that helps the model to learn a distinct representation for them. In traditional methods, to derive a unique representation for an entity such as *"Donald Trump"*, embeddings for two parts of the word have to be averaged, multiplied or transformed in a separate post-processing step, which is not only inefficient but also misleading; as the name *"Donald"* could refer to different people in a different contexts. Grouping multi-word names from the starts helps avoid such problems.

- *Unifying all surface forms of an entity:* Mentions of the same entity with different surface forms are considered a unit identity, e.g., *"Obama"* and *"Mr. President"* in the time frame after 2009 until 2017. This approach lead to more word context pair for each entity, which can increase performance. For example, all the words surrounding *"Obama"* or any other surface form of the entity are considered as context words for *"Barack Obama"*.

- *Unique representation of temporal information:* We identify expressions that are dates or time, and normalise them into a single unique date. Hence, we can incorporate the temporal information as the context of a word, and at the same time learn a vector representation for the dates, which is missing from the traditional word embeddings.

- *Customizable search:* Since entities have pre-defined types, such as actor or location, we can perform a more effective search in the embedding space. The neighbourhood of a word can be filtered to contain only relevant types, which is beneficial for finding relationships between people or organizations.

- *Flexible neighbourhood definition:* In a term-based model, a word is only defined by the terms surrounding it. Annotated named entities, however, permit different definitions of a word, based on the type of entities that surround it. For example, when looking for the nearest neighbours of a word, the neighbourhood can be filtered to contain only entities of a specific type or remove the entities altogether and look only at surrounding terms. We take advantage of this property in Chapter 5 and introduce a new search criterion that succeeds in capturing entity-entity relations better than term-based models.

In general, removing ambiguous mentions and annotating the corpus provides the models with more information and is likely to result in better input features for entity related tasks, such as information retrieval [Diaz et al. 2016; Kuzi et al. 2016], and named entity recognition [Siencnik 2015]. Since there is no framework available that uses such embeddings as an input, the claim for better performance in any downstream task can only be validated if such frameworks are created. However, in this thesis, the focus is not on constructing task-specific features for a downstream task, but rather to create a general-purpose enity-based embedding model that is comparable to normal word embeddings. As a result, we focus only on creating entity embeddings that can perform similarly or even better than normal word embeddings on common term-based evaluation tasks.
In the remainder of this chapter, we introduce methods for jointly training entity and term embeddings on an annotated corpus. The main objective of entity-based models is to upgrade the classical word embeddings by incorporating the type of the word, and also to learn a unique representation for named entities mentioned in the text. We focus on learning these embeddings by tweaking the existing word and graph embedding techniques and evaluating which one works best. For this purpose, we propose two approaches:

1. To naively include entities in our models, we train the well-established word embedding models on a corpus, annotated with named entities.
2. To better capture the entity-entity relations, we take advantage of the graph representation of the corpus, and embed the nodes of co-occurrence graphs extracted from the annotated text.

To enhance the performance of our models, we try a wide range of word and graph embedding techniques and compare them against word embedding models trained on raw text.

3.2 Word Embeddings on Raw Text

State-of-the-art word embedding models are trained on raw text (without entity annotation). The word2vec and GloVe models, explained in Chapter 2, Sections 2.4.1 and 2.4.2 respectively, are chosen as representatives of classical word embeddings. From the word2vec model, the skip-gram architecture is chosen. We refer to the word2vec and GloVe model as rW2V and rGLV, where *"r"* denotes raw text input. The text is cleaned only by common pre-processing steps, such as tokenization and stop word removal. The complete pre-processing steps are explained in Chapter 5, Section 5.2, where we discuss the training data. Since no entities are annotated, all words are treated as terms. These models are used later in Chapter 5 to assess the performance of models proposed in this study.

3.3 Word Embeddings on Annotated Text

Named entities $N \subseteq V$ are a subset of all words in the vocabulary that fall into a pre-defined category. In other words, words that have a specific type are considered as named entities. Hence, the simplest way to obtain their embeddings would be to run the well-established word embedding models on the annotated text, where these named entities are identified and annotated. Our contri-

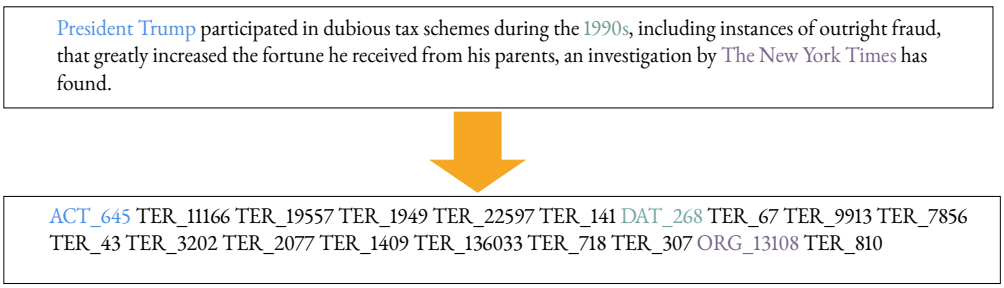


Figure 3.2: An example of the pre-processing for annotated text. All the entity mentions are replaced with their unique identifier and the unnecessary POS-tags are removed.

bution to these models is not the modification of the underlying method, but the input. The input of the model is no longer the raw text, but a corpus with annotated named entities. Annotation involves POS tagging, named entity recognition and disambiguation, which are performed using the state-of-the art tools on raw text. Since word embeddings learn the representation for each token in the text, our definition of the token is changed after annotation. A token is no longer a single word separated by a space but a named entity with a specific type. Any word that does not fit into any of the pre-defined types is considered as a term. Unlike the set of all words $V = N \bigcup T$, in which multiple entities can share a single ambiguous surface form, entities are presented with their unique identifiers. In our case, the identifier is the type of the word and a numeric key, e.g., *ACT_17*, for an actor. As a result, compound words, such as names of people and organisations, are considered as one word. For example, all mentions of *A_17*, which corresponds to the actor *"Barack Obama"* are represented with a single identifier. This process makes the text less ambiguous. Furthermore, depending on the performance of the named entity recognition tool all other indirect mentions, e.g., *"Mr. President"*, *"President of the United States"*, are also replaced with their corresponding entity identifier, resulting in more context pair for a focal word. The same applies to different surface form of dates, as all the dates in their different surface forms are normalized to a single form. Moreover, words that belong to more than one entity type, have an embedding for each type. For example, *"Washington"* can either be a name of a person or a city and has two separate embeddings , one for the actor and on for the location context. The same naming strategy applies for terms, e.g., *TER_22*, where all the unique mentions of terms in text are identified and replaced with their corresponding identifier. By adding the type information in the name of the identifier, we can group named entities of the same type and also to search for a specific type.

Since entity annotation requires POS tagging, we used the POS tags to remove stop-words and punctuation. An example of the text after annotation and pre-processing is shown in Figure 3.2. After annotation, we use word2vec and GloVe and refer to them as aW2V and aGLV, where *"a"* denotes the use of annotated text.

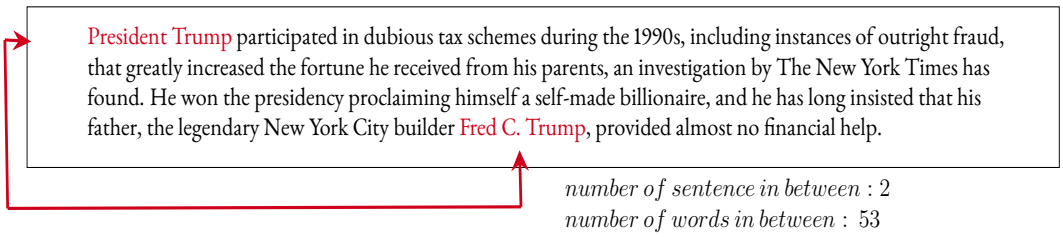


Figure 3.3: An example text to show the relation between two actor entities, *"Donald Trump"* and *"Fred C. Trump"*. If the word-based window size is applied then the relationship between the two entities is not captured, even though they are only one sentence apart.

3.4 Node Embeddings of Co-occurrence Graphs

One motivation for incorporating the graph-based methods to enhance the performance is the additional information that such graphs contain. Most word embedding methods consider a fixed sized window of few words to determine if two words are related to each other. Named entities, however, have more complex relational structures. An actor and an organization can be related even if they are mentioned several sentences apart. An example text is shown in Figure 3.3, where possible relations between two actor entities, *"Donald Trump"* and *"Fred C. Trump"* are shown. If the context window is limited to a few words around the center word, then the relationship would not be detected, except for a window size of over 53 tokens. As a result, word-based windows are unable to detect certain relationships that go beyond a sentence limit. On the other hand, if we look at the sentence distance of even one sentence away, *"Donald Trump"* and *"Fred C. Trump"* would be considered related. This is one of the reasons behind the choice of the LOAD model for the co-occurrence graph. The LOAD model applies a sentence-based window to find entity relations and a word-based window for relations among terms, thus, capturing entity-entity relations better than a word-based window.

We defined a co-occurrence graph of words in Chapter 2, as a graph $G = (V, E)$, where V is the set of words in the vocabulary (nodes) and E is the set of edges. Furthermore, we assume that there are edge weights that encode some notion of distance between the words. We extract such a graph from the textual corpus with named entity annotations. As a result, we obtain a weighted heterogeneous graph $G = (N \bigcup \overline{N}, E)$ of named entities and terms, where the set N contains entities of different types and $T = \overline{N}$ denotes all the words that are not a named entity (terms). For a text with entity annotations in particular, graphs implicitly extracted from text, such as LOAD, can serve as graph representations of the corpus. By using a sentence-window for finding the relationship between entities, they can capture entity relations even sentences apart, while applying a word-based distance for terms. Since these graph represenations capture the relations of all terms and entities in the text, by embedding the nodes, we can obtain word embeddings for both. We later show that graph-based methods achieve a better performance in comparison to aW2V and aGLV on the annotated text. For embedding nodes, we chose DeepWalk (DW) [Perozzi et al. 2014b] and VERSE (VRS) [Tsitsulin et al. 2018a], explained in Chapter 2, Sections 2.6.1 and 2.6.4, respectively.

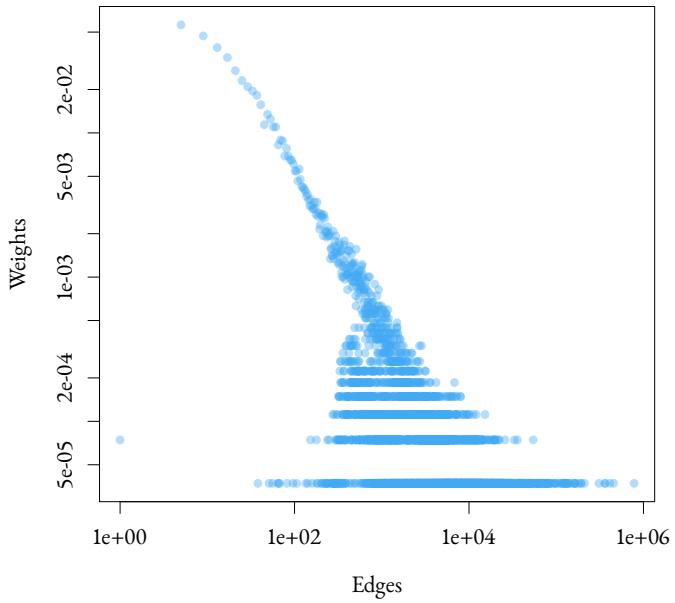


Figure 3.4: Weighted degree distribution of the LOAD network.

Since we work on weighted graphs and weights display the strength of the relationships, we modify the DeepWalk model to take edge weights into account. For this reason, *weighted random walks* are introduced to replace the uniform random walk in the original DeepWalk model. In such a weighted random walk, the probability of a node visiting its neighbours is proportional to the edge weight, where the more stronger the relationship, the more likely a neighbouring node is to be visited. Hence, the probability of visiting node j from vertex i with edge weight $e_{i,j}$, where E_i denotes the set of all edges starting from node v_i, is given in Equation 3.1. f is a normalization function that we will subsequently introduce.

$$P_{i,j} = \frac{f(e_{i,j})}{\sum_{f(e_k) \in E_i} f(e_k)} \tag{3.1}$$

In a word co-occurrence graph, some words co-occur many times and some only a few times. Therefore, the weights can fluctuate between small numbers up to tens of thousands, resulting in highly unbalanced transition probabilities, causing some nodes to never be visited in a random walk. The weighted degree distribution for LOAD graph, extracted from our training corpus, as an example of such unbalanced distribution, is shown in Figure 3.4. The unbalanced distribution may cause the random walker to focus on certain nodes, while ignoring the others, in other words, the edge that have a large weight would be overemphasized, which can lead to poor representation of nodes with smaller weights. To solve this issue and create a more balanced weight distribution, we use different normalization functions on the edge weights:

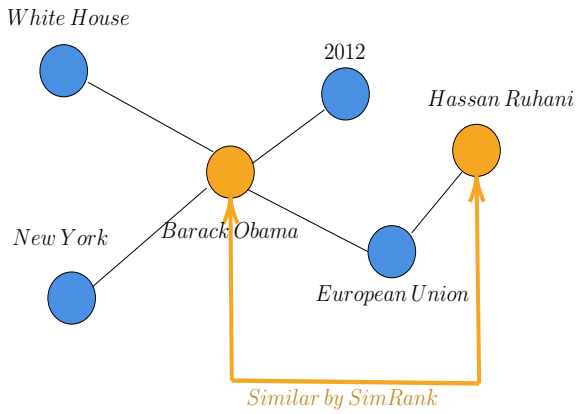


Figure 3.5: An example of SimRank similarity. Although the relationship between the president of Iran and the U.S. is discovered by SimRank, the more important relationships, namely direct neighbours are disregarded. The similarity between *"Barack Obama"* and its direct neighbours are zero.

1. $f = \mathrm{id}$, identity function, where the weights remain unchanged. We refer to these models as DW_{id}, where id refers to the identity mapping.

2. $f = \log$, logarithmic normalization, where all edge weights are replaced by their logarithm and the transition probabilities are computed accordingly. These models are denoted by DW_{log}.

Initially, we also experimented with the square root ($f = \mathrm{sqr}$), but we found that it produces similar results to the logarithmic normalization, so it is removed from the results.

Our next candidate for generating node embedding is the VERSE model. Unfortunately, The VERSE model does not support the incorporation of the edge weights, and the edge weights are disregard for this model. From the three similarity measures that VERSE offers, we chose Adjacency Similarity, explained in Chapter 2 in Section 2.6.4, to produce the entity embeddings. Our reason is two-fold: first, the PPR, explained in Chapter 2 in Section 2.6.4, relates to the stationary distribution of a random walk with restart and we already represented the random walk-based models with DeepWalk. Second, SimRank, explained in Chapter 2 in Section 2.6.4, is a measure of structural relatedness, between two nodes, based on the assumption that two nodes are similar if they are connected to other similar nodes. Although words that co-occur often with the same words can themselves be considered similar, if we only look at nodes that have common neighbours we lose most of the adjacency relations in the graph. An example is illustrated in Figure 3.5. Since the SimRank of two nodes with no common neighbour is zero, the direct neighbours are disregarded. However, SimRank tends to discover interesting relationships, such as the one shown in Figure 3.5 and possibly if a new similarity measure is introduced that creates a trade-off between SimRank and Adjacency Similarity, the results might improve. For this study, however, we choose only Adjacency Similarity to learn the embeddings and we leave the development of a new similarity measure as future work. Since Adjacency Similarity is based on the normalized adjacency matrix, it captures the neighbourhood relations and focuses on the direct neighbours of a node.

The other two embedding methods explained in Chapter 2 are not used to generate entity-based embeddings. We could not use the LINE model for the following reasons : First, because of the unbalanced weight distribution, the optimisation objective for the second-order proximity of the LINE model becomes ill-defined. Hence, one part of the embedding for this model cannot be trained properly. Second, the Adjacency Similarity of VERSE model correlates with the first-order proximity in LINE and can be used as representative of first-order proximity. The node2vec model is quite slow and inefficient for large graphs, consequently, because of the time and memory overhead fora co-occurrence graph extracted from a large corpus of text, we did not use this method.

3.5 Summary of Entity and Word Embeddings

In this chapter, we introduced two methods to extract entity and term embeddings from an annotated corpus. One method learns embeddings directly from the text, annotated using state-of-the-art named entity recognition and disambiguation tools. While the other generates a co-occurrence graph for the annotated corpus and embeds its nodes. In the next chapter, we attempt to create embeddings separable by entity type, namely faceted embeddings, and in in Chapter 5, we report the evaluation results for both entity-based models and faceted embeddings. Furthermore, we show why the extraction of a co-occurrence graph is crucial for better performance.

4 Faceted Embedding Model

To further understand the effect of each entity type on the meaning of a word and as an effort to make the dimension of entity embeddings more interpretable, we experiment with creating embeddings with separable parts. By studying these models we gain some insights about the significance of each entity type for vector representations of words and explore if an embedding can be efficiently divided based on these types. In order to create these faceted models, we modify the well-established word- and graph-embedding techniques. In Section 4.1, an overview of the objectives is given. In Section 4.2, the first model based on the GloVe model is proposed. For the faceted GloVe, we experiment with different cost functions, to obtain good performance. In Section 4.3, a faceted model based on word2vec is introduced, which takes the edges of a co-occurrence graph as input. Since the graph-based embeddings have the best performance for entity-based embeddings, we experiment with DeepWalk to generate similar embeddings for the faceted model in Section 4.4. For all models, the modifications to the original techniques and training procedures are discussed.

4.1 Overview and Objectives

Word embedding are in general very ambiguous, it is unknown what each value in the vector representation indicate, if each value corresponds to a distinct attribute of the word or shows to what degree each word belongs to a certain group. We introduce faceted embedding models to address this problems, where we divide the word embedding into components that show a pre-defined attribute of each word, namely its relation to entities with specific type. Since we are looking at entity-based relations, the corpus should be annotated with named entities. A general pipeline of the approach is shown in Figure 4.1, where the entity recognition and disambiguation is applied to the input text that is used to extract a co-occurrence graph. The graph representation of the corpus is then used by graph-based embeddings or variations of common word embedding techniques to generate a typed

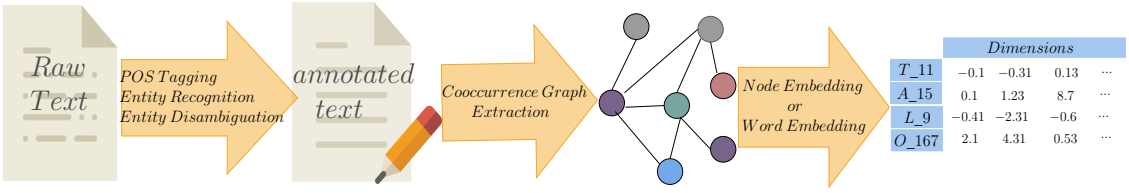


Figure 4.1: Pipeline for generating faceted embeddings. The first step is the annotation of the raw text with POS tagging, entity recognition, and disambiguation. A co-occurrence graph is extracted, which is used as input for node embedding or variation of word embedding methods.

Figure 4.2: Faceted embedding of the entity *"Donald Trump"* for a short paragraph. Each part of the embedding corresponds to the relation of the center word to that specific type. Each entity type is illustrated with matching color in text and embedding.

embedding, in which instead of capturing the semantic of a word in relation to the entire vocabulary (as normal word embeddings do), we capture the semantic in relation to entities of a specific type. The entity-entity relations are very important for the generation of faceted models, hence, unlike the entity-based embedding in Chapter 3, applying the normal word embeddings with a word-based window would not capture the relations that these models need to reflect and extraction of a co-occurrence is graph is a mandatory step. This process is repeated for all entity types available for the model, which results in multiple embeddings, each specific to a type. These embeddings can be concatenated to generate a final embedding or used separately based on the use-case scenario.
Ideally, a faceted model is able to capture surrounding relations for each type separately in different components of the vector, where the relation to entities that co-occur with a word would be captured by the corresponding component. An illustration of a faceted embedding for the entity *"Donald Trump"* is shown in Figure 4.2. The center word is the entity *"Donald Trump"* marked in red and each type is annotated with a different color in the text. Any word that is not an entity or a date is considered a term. Each component is responsible for mapping the entity *"Donald Trump"* in the neighbourhood of entities of a specific type that co-occur with it.
Some of the potential advantages and motivations for analysing each part separately are as follows:

- *Interpretability of relations:* Since each part of the embedding vector encodes relations to one of the entity types, e.g., the actors part only encodes the actors in the context of a word. Through these separate components, the relations between two words or entities become more interpretable. Any similarity measure to compare two word vectors can be applied to each component separately. Thus, the similarity between words can be broken down into their different attributes. If two entities are closer in location space and farther in actor space, it is possible that they are situated close locally, but are not related to the same people.

- *Change in meaning:* Exploring and visualizing different components of an embedding over time gives us insights into how that entity has evolved over time, with respect to that component. When applied to a corpus with a temporal aspect, like news articles, faceted models can

illustrate how a word changes its meaning in relation to different types. For example, *"Donald Trump"* is mentioned frequently in location context with *"Iran"* during the discussion for the nuclear deal, but some time after the abandonment of the deal the topic fades away and *"Donald Trump"* is mentioned more often in relation to other countries. This information can be extracted from the location subspace of the word embedding. Consequently, the values for the location part of the embedding will bring the word vector closer to any other entity that is also frequently mentioned with the Iran deal during the nuclear discussions, e.g., *"Khamenei"* (Iran's supreme leader). Even if *"Khamenei"* is not directly mentioned with *"Donald Trump"*, they are mentioned in the same context location-wise, and, therefore, are mapped to points close to each other in the location domain. If embeddings are reconstructed for a time span after the nuclear abandonment, the two mentioned entities should become less similar in location aspect. Although the same experiment can be done on normal embeddings, where all the words are treated equally, the distinguishable components give us additional insights as to what aspect has invoked the change.

- *Interpretability of evaluation tasks:* Separable parts also play a role in interpretability of evaluation tasks. We experiment with using only a specific component in our evaluations, in Chapter 5, to find out which type of entity surrounding a word is more influential for a certain task.

- *Flexible neighbourhood search:* With faceted models, the search for a similar word or entity becomes more flexible. Same as the embedding space, the search space can be divided into various types, where one can look for neighbours in a specific context. The type-specific information can be used in the search, where one can query for entities that are closer to a certain word in the temporal or location aspect. For example, *"Washington"* as a political person should appear more often with the same actors and has a different actor dimension than the city of *"Washington DC"*. The standard embedding treats all words equally and therefore, cannot reflect this type of similarity. Since the component for different types are independent and can be combined in an arbitrary way, a combination of different components can tailor search results further and create a type-specific search.

To achieve this type of embeddings, we define faceted embeddings as word embeddings, where each dimension represents the relation of the embedded word to a specific type of entities surrounding it. Which types are considered during training is arbitrary, but for the purpose of this work, we choose to contain actors, locations, organisations, dates, and terms, since these types are available by the LOAD model. An example of such an embedding vector (w) for a corpus containing types of actor (ACT), location (LOC), organisation (ORG), date (DAT) and term (TER) is shown below:

$$w = \Big[\underbrace{a_{1,1}...a_{1,M}}_{\text{ACT}} \,|\, \underbrace{a_{1,M+1}...a_{1,2M}}_{\text{LOC}} \,|\, \underbrace{a_{1,2M+1}...a_{1,3M}}_{\text{ORG}} \,|\, \underbrace{a_{1,3M+1}...a_{1,4M}}_{\text{DAT}} \,|\, \underbrace{a_{14M+1}...a_{1,5M}}_{\text{TER}}\Big] \tag{4.1}$$

Provided that the most common dimension for word embeddings are between 100 to 300, to maintain the same order of dimension in faceted embeddings, all the different parts have the same dimension but in a lower magnitude (20 to 50). These small embeddings are concatenated to create the final embedding, where each part is independent of the rest and is usually trained separately.
With these independent components, the embeddings stand to be more interpretable. Although a model that divides the embedding space into entity types has not been studied, some work has been done to make the vector representations more interpretable. In the work of Frauqui et al. transformation of word vectors into sparse (and optionally binary) vectors is proposed [Faruqui et al. 2015]. Each vector is projected into an over-complete binary vector, where each dimension represents a feature similar to ones used in traditional NLP systems but found automatically during training. However, since their dimensions are binary valued, there is no notion of the extent to which a word participates in a particular dimension. Later, a model based on rotating the word vectors was introduced in order to improve the interpretability. [Park et al. 2017]. Recently, a neural network-based approach to the problem was introduced by using sparse auto-encoders [Subramanian et al. 2018]. Although these works shine some light on the potential meaning of each dimension, they focus purely on terms and share the problems of all term-based models. whereas, we generate embeddings that each part reflects the word relation to entities of a certain type and is based not only on terms but entities of different types.
As explained in Chapter 2, the aim of the word embedding is to reduce the high dimensional space of text and embed words into a meaningful space, where words with similar meaning are mapped to points close to each other. In the case of faceted embeddings, it is as if we are dividing the textual space into all possible entity types, where each subspace contains only entities of a specific type. To put it differently, the space of all words V is represented as $V = A \bigcup L \bigcup O \bigcup D \bigcup \overline{N}$, where A,L,O,D are the sets of all actors, locations, organisations and, dates, respectively. The set $T = \overline{N}$ denotes all the words which are not a named entity (terms). The goal is to map each word in the vocabulary into all the subspaces, where words that co-occur with the same entities of a specific type are mapped closer together in that particular subspace. Since all the subspaces are independent of each other (e.g., the space of all actors does not have any of the location entities inside it), each embedding learned on the different subspace is also independent. Hence, it is perfectly reasonable that two words are close in one subspace and far in another. The independence of type also allows for arbitrary combination and independent analysis of learned components after training.
In the remainder of this chapter, we aim to introduce faceted models as a general framework for learning embeddings with known types and separable components. To create such embeddings, we propose three approaches, two of which are based on word embedding models and one use the graph embedding techniques. It is worth noting that while we used word embedding methods, the models are modified to take a graph structure as input rather than raw text. Our three models are constructed as follows:

1. By modifying the cost function of the GloVe model to train a separate embedding for each entity type on an annotated corpus. Although the original GloVe uses the word co-occurrence matrix as input, we transform the model to use the adjacency matrix of a co-occurrence graph.
2. By using a variation of the word2vec model, which supports an arbitrary definition of context for each word to train an embedding for each entity type.
3. By Modifying the graph embedding model DeepWalk to generate embeddings based on neighbours of a specific type.

4.2 Faceted GloVe

In the previous chapter, we indicated that the LOAD edge weight captures the corpus relations similar to the co-occurrence matrix and that meaning components can be extracted from them. In this chapter, we use this knowledge to create faceted embeddings from the weighted adjacency matrix of the LOAD graph. In the first section, the construction of a weighted adjacency matrix is explained followed by the model definition. We denote the faceted GloVe model by *f*GLV, where *"f"* indicates the faceted model. The proposed model contains two type of cost functions, that are discussed separately in Sections 4.2.2.1 and 4.2.2.2.
The complete training task can be formulated in four steps:

1. Extraction of co-occurrence matrices for different types, explained in Section 4.2.1.

2. For each type, a GloVe based embedding is learned using either a separate or unified cost function, discussed separately in Section 4.2.2.1 and 4.2.2.2.

3. The context embedding is kept and the focal embedding is either disregarded or added with focal addition, discussed in Section 4.2.2.

4. The results of the third step are different embeddings for each type, which can be concatenated to generate the final faceted embedding.

4.2.1 Weighted adjacency matrix

To learn embeddings, a weighted adjacency matrix for each type of entity and term is needed (ACT, LOC, TER, ORG, DAT). Matrices are constructed based on the weighted edge list of the co-occurrence graph, where the weights can be co-occurrence counts (identical to the GloVe model) or, any other distance measure defined by the graph. For example, the weighted adjacency matrix for actors contains the edges that have an actor as start nodes and any other type as the end node. Since in this thesis we use undirected co-occurrence graphs, the same edge exists in the opposite direction. For instance, an edge between an actor and a term is repeated twice and will be used once to create the actor matrix and once to create the term matrix. A visual construction of the matrices is shown in Figure 4.3, where the entries of the matrices are edge weights between words. The columns marked

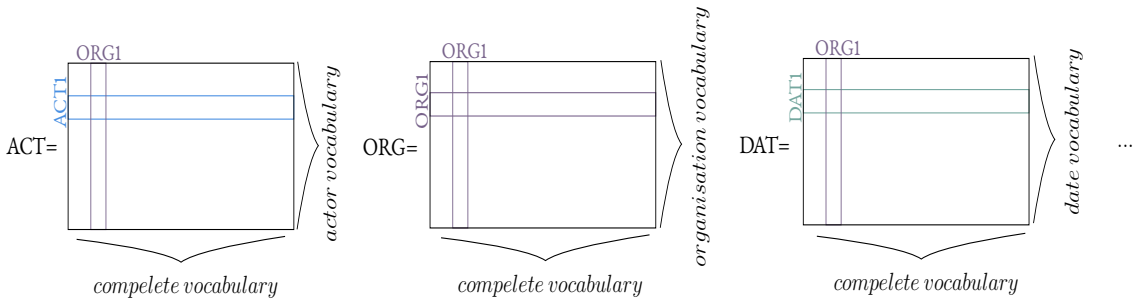


Figure 4.3: Faceted weighted adjacency matrix. For each type of entity a weighted adjacency matrix is created, which contains all entities of that type on rows and all words in the vocabulary in columns.

in blue in matrices $\mathrm{ACT} \in R^{|V_{\mathrm{ACT}}|\times|V|}$, $ORG \in R^{|V_{\mathrm{ORG}}|\times|V|}$ and $\mathrm{DAT} \in R^{|V_{\mathrm{DAT}}|\times|V|}$ are used to learn the actor, organisation, and date part of the ORG1embedding, respectively. The complete vocabulary is divided into words belonging into each entity type. As a result, if V is the complete vocabulary and A,L O, D, and T are the set of all actors, location, organization, dates, and terms in the graph, we will have $V = A \bigcup L \bigcup O \bigcup D \bigcup T$. The model will learn embeddings for all entities in the row and all entities in the column of the matrix. In case of a symmetric co-occurrence matrix, the model learns the same embedding twice. Nevertheless, in our case the matrices are not symmetric, because the size of the vocabulary for each entity type differs. This asymmetry leads to a definition of a new cost functions that are explained in following sections.

4.2.2 Cost function of faceted embeddings

For each different adjacency matrix, different embeddings need to be learned. To train all embeddings, two methods are proposed. First, training a network for each type separately and concatenating the outputs to generate the faceted embeddings. Second, train all the embeddings at once in a single network with a unified cost function. We refer to separate and unified cost function as $f\mathrm{GLV}_{sep}$ and $f\mathrm{GLV}_{uni}$, respectively. the second method, despite the compactness, is slower than optimising separate cost functions. It is worth noting that the switching between the training method has no effect on the quality of embeddings, only the training time. For the sake of completeness however, we present both methods. Below we consider some alteration to the cost function of the original GloVe, which are considered to make faceted embedding possible. Below we list the modifications needed for the both cost functions:

- *Asymmetry of context and focal embeddings:* Similar to the GloVe model, embeddings for both context and focal words are learned, but the focal embedding can no longer be naively summed up with the context. Instead, the focal embeddings can either be disregarded completely or added in a new way. For this purpose, we propose *selectional addition*, where the focal embedding of a certain type is only added to the corresponding embedding in the context. As an example, take the adjacency matrices in Section 4.2.1 as our input, then for the matrix ACT, focal embedding will learn to encode each row, while the context will encode the columns. The rows of the matrix will represent embeddings of all the actors with respect

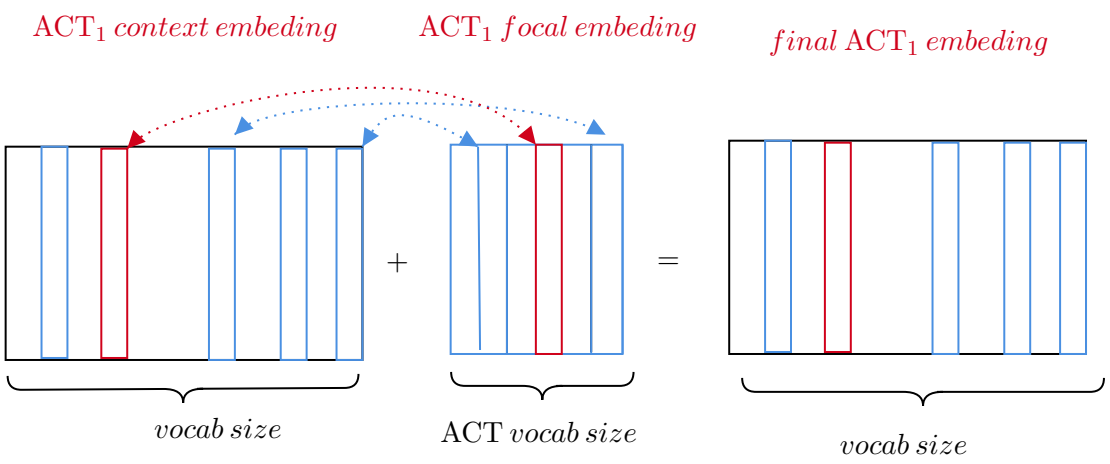


Figure 4.4: Selectional addition of focal and context embedding in case of actor embeddings. ACT$_1$ is shown as an example, where embeddings in focal and context matrices are combined to make the final embedding. Focal embeddings are only added to the corresponding actors in the context matrix.

to the whole vocabulary. We are interested in column embeddings that encode the actor part for all the words in our vocabulary, which is the purpose of this model. Focal embedding also contains information about the whole vocabulary, but only for the focal words (actors), adding back this information in terms of selectional addition is beneficial to the model. The selectional addition adds the focal embedding of a type only to the respective entities of that type in the context matrix. The rest of embeddings remain intact, because a corresponding embedding column for them in the focal matrix does not exist. A visualization of the focal addition for type actor can be seen in Figure 4.4.

- *Final concatenation:* The output of the model is separate embeddings for each entity type. These vectors can later be combined to generate the complete faceted embedding. As the embeddings are independent, the combination can be arbitrary with only the desired components kept.

- *Weighting function:* The weighting function is the weighting function for the GloVe model. The faceted model is valid with or without the use of a weighting function, but since the weights of a co-occurrence graph are highly unbalanced, without a weighting function, very large weights will overpower the smaller ones and reduce performance. Therefore cutting off the weights at a certain threshold can to some extent balance out this effect. The hyperparameters of the weighting function should be tuned based on the dataset.

- *Normalization:* In an attempt to boost performance, weights of the graph were normalized using logarithmic normalization. With normalization, we reduce the range of values, in which the weights fluctuate. Moreover, the gradient descent algorithm can converge more smoothly to the functions minimum. Unfortunately, adding the normalization for the GloVe model

made the problem ill-conditioned, so the algorithm could not converge to a solution and, therefore, is dismissed. The reason for this could be that the logarithmic normalization might result in negative values for values between 0 and 1. In the cost function, a second log is applied to the inputs, and since the log of a negative number is undefined, it would result in undefined gradient updates and untrainable weights. Other than log normalization, a linear transformation like *min-max normalization* was also experimented. Min-max normalization forces all weights to be between 0 and 1 and is calculated with Equation 4.2, where $e = (e_1, ..., e_n)$ is the set of input weights and z_i are the ith normalized outputs. Unfortunately, adding this form of normalization does not improve the results and is also removed from the final model.

$$z_i = \frac{e_i - min(e)}{max(e) - min(e)} \tag{4.2}$$

- *Adding 1 to the logarithm:* The cost function applies a logarithm to the weights. For an edge weight of 1 the $\log(1) = 0$, but the weight of 1 should not indicate that there is no relationship between them. Therefore, we tested the model with the addition of 1 to the log of weights, to adjust for this effect. With this change, the cost function of the GloVe model is changed to Equation 4.3. Although in some cases this change improved the model performance, in other cases it generated poor results and was also removed from the model.

$$J_e = \sum_{j=1}^{|V|} \sum_{i=1}^{|V_f|} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b_j} - (logX_{ij} + 1))^2 \tag{4.3}$$

All the discussed alterations to the original cost function of the GloVe can be applied to both unified and separate cost function. In the following, we discuss the two cost functions for the faceted GloVe and explain their difference.

4.2.2.1 Separate cost functions

The general cost function for a single network is the one used for the GloVe model with minor modifications. As noted in the previous section, context and focal embeddings are no longer symmetric, since the size of context vocabulary is different from the focal. By limiting the focal words to a certain type we try to infer the impact of that type on all the words and generate type-specific embeddings. Accordingly, the context embedding in each case reflects these type-specific properties for all words. Biases for focal and context embeddings are longer be symmetric as well. Therefore, the GloVe cost function is re-written to Equation 4.4, where V_f is the size of the focal vocabulary. The function f is the weighting function, which is applied to the edge weight e_{ij} between nodes i and j. w_i is focal embedding for the start node and w_j is the context embedding for the end node.

$$J_e = \sum_{j=1}^{|V|} \sum_{i=1}^{|V_f|} f(e_{ij})(w_i^\top \tilde{w}_j + b_i + \tilde{b_j} - loge_{ij})^2 \tag{4.4}$$

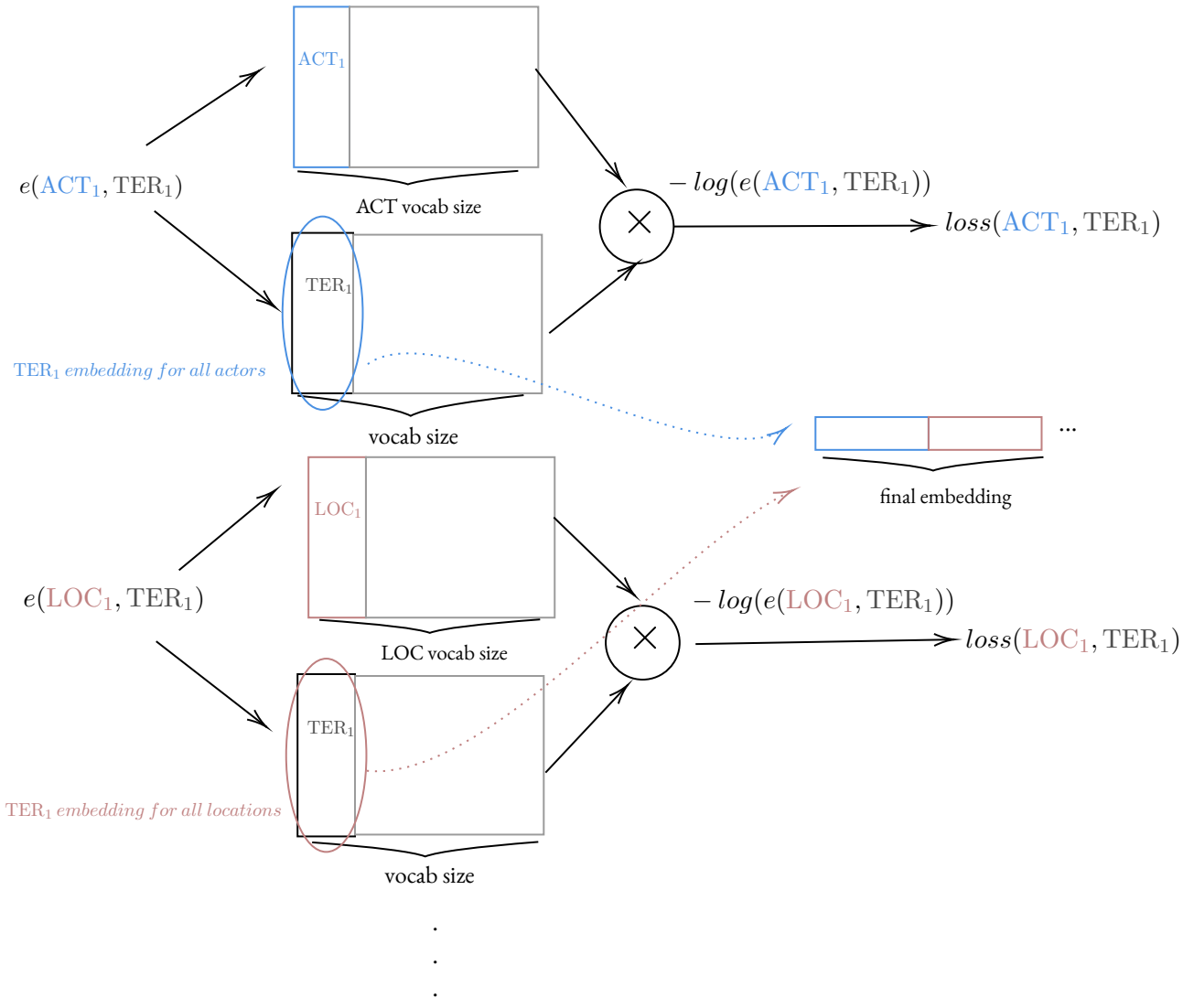


Figure 4.5: Faceted embedding with separate cost functions. Contains training of five different networks (two are shown in the figure) for a single input. The target word is TER_1, which is being trained against all actors, organisations, locations, dates and terms. The final embedding is created by concatenating the columns in context embedding matrices that are associated with TER_1.

Equation 4.4, minimizes the distance between the dot product of embeddings for start and end node of an edge with log of their edge weight. This cost function has to be minimized separately for each entity type, where the focal vocabulary changes to reflect the type being learned. Thus, the training will contain a separate training phases for each type and a final concatenation.

The Figure 4.5, shows visual illustration of training with separate cost function. Two networks are shown as an example, where a single edge affects embeddings of the type associated with the focal (start node), while the edge in the opposite direction affects the embedding associated with the context (end node). Each network is trained separately and a final embedding concatenation of all context embeddings of different types.

4.2.2.2 Unified cost function

The unified cost function is simply the summation of all the separate cost functions in the previous section, where K is the set of all possible types, shown in Equation 4.5.

$$J = \sum_{k=1}^{|K|} J_k = \sum_{k=1}^{|K|} \sum_{j=1}^{|V|} \sum_{i=1}^{|V_f|} f(e_{ij})(w_i^\top \tilde{w_j} + b_i + \tilde{b_j} - log e_{ij})^2 \tag{4.5}$$

As a result, all embeddings can be generated with a single network. For a better understanding of the difference between the methods a visual comparison is shown in Figures 4.5 and 4.6 for a loss on a single input. Each input is an edge with its corresponding weight, if the input has the starting node of type actor, the rest of the layers related to other types of embeddings will be frozen during training and only the weights for the actor network will be updated. Hence, it is as if all the networks were trained separately.
In Figure 4.6, we show an illustration of training the unified cost function. As a result of unification of the five different cost functions, one network with five times more parameters is trained, where each input edge only affects weights of the network associated with the focal word (start node). Therefore, an edge between an actor and term does not affect location related embeddings. Since edges are undirected, the edge in the opposite direction will be fed as an input to the term network and hence, no information is lost.

4.2.3 Vectorized faceted embeddings

In order to speed up the model and use batch gradient descent we need to vectorize the cost function. In the following, we go through the vectorization process for embeddings with separate costs. The same idea can also be applied during the training of the unified cost function. If we consider W as the matrix of all focal embeddings and $\tilde{W}$ as the matrix of all context embeddings stacked row-wise, then we can vectorize the model according to Equation 4.6.

$$J = (W\hat{W}^\top + B + \tilde{B} - logX)^2 \tag{4.6}$$

Unlike the original GloVe model, dimensions of W and $\tilde{W}$ are no longer equal as our weighted adjacency matrix is no longer symmetric $X \in R^{|V_f| \times |V|}$. V_f is the set of all focal words, which is defined by the input. If actors are the focal words then V_f is the set of all actors in the vocabulary. Parameters of the model are as follows:

- $W \in R^{|V_f| \times M}$ for focal embeddings.
- $\tilde{W} \in R^{|V| \times M}$ for context embeddings.
- $B, \tilde{B} \in R^{|V_f| \times 1}$ biases for focal and context embeddings.
- $X \in R^{|V_f| \times |V|}$ co-occurrence matrix of the whole vocabulary.

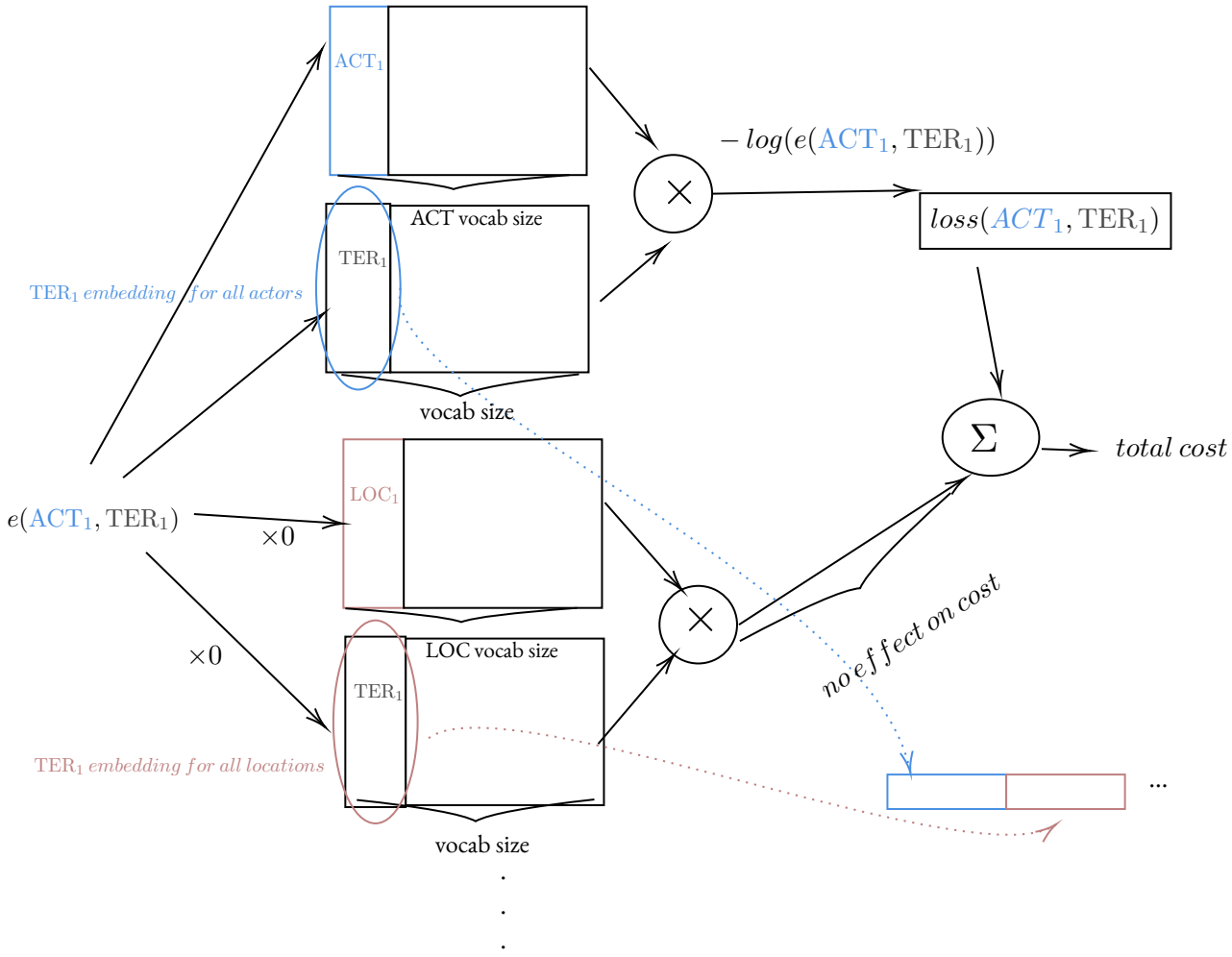


Figure 4.6: Faceted embedding with unified cost function. Training for a single input is shown using only one network. The target word is TER_1, which is being trained against all actors, organisations, locations, dates and terms. While training, weights related to actors are only updated if the input edge weight has the start node type of an actor. Meanwhile, rest of the weights related to other entity-types are freezed. The final embedding is created by concatenating columns in context embedding matrices that are associated with TER_1.

Biases have to be added row-wise to matrices to preserve the original cost function 2.12. Consequently, biases are transformed to column vectors in the size of the focal vocabulary. A comparison between parameter matrices of the original GloVe model (J_g) and faceted cost function (J_f) is shown below, where the size of the vocabulary is indicated by $|V| = v$ and size of the focal vocabulary by $|V_f| = v_f$.

$$J_g = \overset{W(v \times M)}{\begin{bmatrix} a_{11} & a_{12} \cdots a_{1M} \\ a_{21} & a_{22} \cdots a_{2M} \\ \vdots & \vdots \ddots \vdots \\ a_{v1} & a_{v2} \cdots a_{vM} \end{bmatrix}} \cdot \overset{\tilde{W}^T(M \times v)}{\begin{bmatrix} a_{11} & a_{12} \cdots a_{1v} \\ a_{21} & a_{22} \cdots a_{2v} \\ \vdots & \vdots \ddots \vdots \\ a_{M1} & a_{M2} \cdots a_{Mv} \end{bmatrix}} + \overset{B(v \times 1)}{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_v \end{bmatrix}} + \overset{B|\tilde{V}| \times 1)}{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_v \end{bmatrix}} - \overset{logX(v \times v)}{\begin{bmatrix} a_{11} & a_{12} \cdots a_{1v} \\ a_{21} & a_{22} \cdots a_{2v} \\ \vdots & \vdots \ddots \vdots \\ a_{v1} & a_{v2} \cdots a_{vv} \end{bmatrix}}$$

$$J_f = \overset{W(v_f \times M)}{\begin{bmatrix} a_{11} & a_{12} \cdots a_{1M} \\ a_{21} & a_{22} \cdots a_{2M} \\ \vdots & \vdots \ddots \vdots \\ a_{v1} & a_{v2} \cdots a_{v_e M} \end{bmatrix}} \cdot \overset{\tilde{W}^T(M \times |V|)}{\begin{bmatrix} a_{11} & a_{12} \cdots a_{1v} \\ a_{21} & a_{22} \cdots a_{2v} \\ \vdots & \vdots \ddots \vdots \\ a_{M1} & a_{M2} \cdots a_{Mv} \end{bmatrix}} + \overset{B(v_f \times 1)}{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{v_f} \end{bmatrix}} + \overset{\tilde{B}(v_f \times 1)}{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{v_f} \end{bmatrix}} - \overset{logX(v_f \times v)}{\begin{bmatrix} a_{11} & a_{12} \cdots a_{1v} \\ a_{21} & a_{22} \cdots a_{2v} \\ \vdots & \vdots \ddots \vdots \\ a_{v1} & a_{v2} \cdots a_{v_f v} \end{bmatrix}}$$

4.3 Faceted Word2vec

Word embeddings are used as input features in many NLP tasks. Although methods like word2vec capture semantic features well, they often lack task-specific features. Thus, many studies focus on modifying and tweaking the existing methods for certain tasks, such as text classification [Liu et al. 2018], semantic relation classification [Hashimoto et al. 2015] and dependency parsing [Bansal et al. 2014]. Because with faceted embedding we also aim to modify existing methods to match our specific task, we looked at a wide range of litreture on modifications of word embedding methods to match a specific context definition. In 2014, Levy and Goldberg proposed a method to generalize the skip-gram achitecure of word2vec to include arbitrary contexts and used it to create dependency based embeddings [Levy and Goldberg 2014b]. We use the idea behind this model to define our own context for the skip-gram model, which allows us to learn separate components of the faceted embedding. In this section, first, we give a brief description of embeddings with arbitrary contexts by Levy and Goldberg. Then, we look at our definition of context and modifications to the original model. Finally, we use this knowledge to generate the algorithm for faceted word2vec. The model proposed by Levy and Goldberg uses the skip-gram architecture, thus, we also generate the faceted word2vec based on skip-gram and do not use the CBOW architecture.
For creating the faceted model using embeddings with arbitrary contexts, we follow these four steps:

1. Extraction of type-specific edge lists from the co-occurrence graph, discussed in Section 4.3.2.

2. Training a low dimensional embedding for each type-specific edge list using the method proposed by Levy and Goldberg, where focal context pairs are edges of the graph [Levy and Goldberg 2014b], discussed in Section 4.3.1.

3. Disregarding focal embeddings and keeping only context embeddings, as it represents the relation of a word to entities of a specific type.

4. Concatenating context embedding of all types to create the final embedding.

4.3.1 Embeddings with arbitrary contexts

The general principle behind skip-gram is to bring words that appear in similar contexts in the text, closer in embedding space, where the contexts of a word are the words surrounding it. Therefore, the context vocabulary C is identical to the complete word vocabulary V. In other words, the focal words and their context share the same vocabulary. Nonetheless, for embeddings with arbitrary contexts, this restriction is not required. The context does not need to correspond to words and can be defined based on the use-case, which is an important attribute for the faceted model. The negative sampling objective of word2vec, shown in Equation 2.10, assumes a dataset Q of focal and context pairs (f, c) from a large body of text and samples negative examples form $\overline{Q}$ (the noise distribution). The model assigns a low score to the random negative samples and a high score to the real focal and context pairs. Since the input of the model is word pairs, it can be constructed in an arbitrary way. Levy and Goldberg specifically look at contexts, based on the syntactic relations, where the context is defined by the type of the dependency relation between the head and the modifier. They scan the text once and create focal and context pairs based on specific dependency relations and then use the generated pairs as positive inputs. For each $(f, c) \in Q$, n samples $(f, c_1), \ldots, (f, c_n)$ are constructed as negative examples, where c_j is drawn according to its unigram distribution raised to the power of $\frac{3}{4}$ [Levy and Goldberg 2014b]. Their framework is not only limited to dependency relations but also allows for various context definitions [Levy and Goldberg 2014a]. Following their example, we can define a type-specific context to generate faceted embeddings. We denoted the faceted word2vec model by fW2V, where f indicates the faceted model. In the following section, we show how the edges of co-occurrence graphs are used as focal and context pairs for our study.

4.3.2 Edges of co-occurrence graphs as focal and context pairs

We take advantage of the definition of arbitrary context and define our context based on the types of entities surrounding a word. Our goal is to create embeddings, such that each part corresponds to the relation of the focal word to the specific type of entities as context. A co-occurrence graph extracted from annotated text can help us achieve just that. To obtain each componnent of the embedding we define a type-specific edge list, in a similar fashion as type-specific adjacency matrices for faceted GloVe, where the start nodes are entities of a specific type and end nodes can be any term or entity from the vocabulary. More formally, in the heterogeneous graph $G = (V, E)$, where

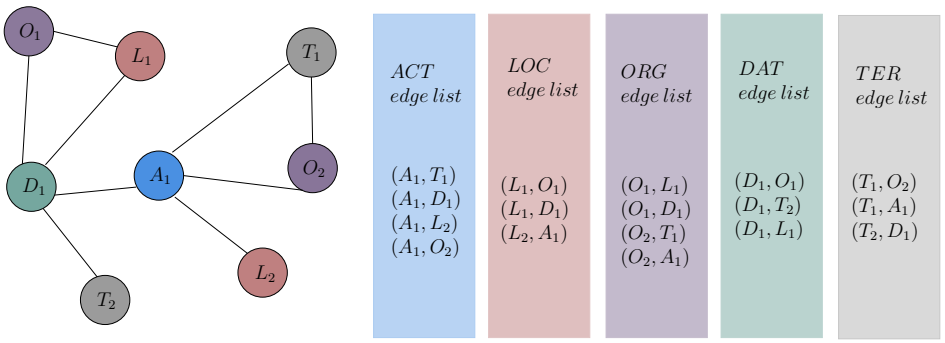


Figure 4.7: An example of creating type specific edge lists for a graph with node type: actor (A), location (L), organisation (L), date (D), and term (T).

$V = A \bigcup L \bigcup \bigcup O \bigcup D \bigcup T$ is the set of all nodes with types actors, locations, organisations, dates, and any non-entity word (term) and E is the set of all weighted edges, we extract five edge lists where the start node is one of the types in set V. An example of edge lists is shown in Figure 4.7. As demonstrated in Figure 4.7, we have a different edge list for each type that are treated as a focal and context pairs (f, c) for the input of skip-gram with arbitrary contexts. Context word can be any node $c \in V$ and focal word f has specific type, e.g, $f \in A$ for generating actor component of the embedding. Thus, the focal and context vocabulary are not equal, as the focal vocabulary is always a subset of the context ($A \subset V$). Based on each edge list the model learns a separate embedding, with entities of a certain type as focal words and the rest of the vocabulary as context. As a result, each context embedding encodes the relation of the complete vocabulary with a certain type. For example, if we look at the context embedding for actors, it encodes the relation of the words in the vocabulary to all the actors. Naturally, if a word does not have an edge to any actor it will not be present in the context embeddings. For such cases, it makes sense to set the actor component of that word equal to a vector of zeros. For example, in Figure 4.7, T_2 will only have a non-zero date component and the rest are set to zero. This approach has some drawbacks, as the cosine similarity between a vector of zero and any other vector is undefined. Therefore, if a component of a vector is set to zero, that word becomes incomparable to all other words in that sub-space. It also causes problems for clustering algorithms that require a distance measure between the points, if the cosine similarity is undefined, then we are unable to compare the words and form clusters. To solve this issue, we add a small number 0.0001 to the vectors of all zero, mapping all the words to the same point, while keeping the component small. Moreover, clustering algorithms are able to compute the distance between the word vectors in the sub-spaces. This approach also has some unwanted implications, if components of two words are identical, their cosine similarity is always one. Thus, any two words that do not have relations to entities of a particular type are considered similar in that subspace, which is not always the case. For example, in Figure 4.7, T_2 and L_2 both have the same organisation component, as they are not connected to any organisation node. The absence of an edge to an organisation does not necessary mean that T_2 and L_2 should be similar in organisation space, which is what the model would imply.

4.4 Faceted DeepWalk

Another way to achieve the faceted model is by embedding the nodes of a co-occurrence matrix, where node embedding algorithm preserves the neighbourhood of a specific For this purpose, we alter the DeepWalk model to meet our needs, and introduce a *type-restricted random walk*.
In 2017, *metapath2vec* was proposed by Dong et al. to study representation learning in heterogeneous graphs with multiple edge and node types, such as coauthor relationships graphs [Dong et al. 2017]. Metapath2vec learns a low dimensional representation of the graph using random walks that are restricted to the only transition between particular types of nodes and edges. They also introduce a heterogeneous skip-gram which maximizes the probability of having the heterogeneous context for a given node [Cai et al. 2018]. For learning the faceted embeddings, however, we take a more straightforward approach. We take advantage of the weighted random walks used in Chapter 3 to generate entity embeddings, to introduce a new type-restricted random walk. With the help of this customized random walk, we restrict the context of a word to certain entity type. In general, faceted DeepWalk model consists of three steps:

1. Generating type specific corpus based on type-restricted fixed length random walks starting from all the nodes in the graphs.

2. Applying the skip-gram model on the random walk corpora.

3. Concatenating the resulting embedding for each type to achieve the final faceted model.

Type-restricted random walk: Formally, a type-restricted random walk in graph $G = (V, E)$ is denoted in the form of $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v_n$, where the nodes v_i belong to a certain node type. In each step, the random walker is only allowed to visit the next node if it satisfies the type restriction. For example, in a graph, where $V = A \bigcup L \bigcup O \bigcup D \bigcup T$, to sample only locations related to a node, we perform a location-restricted random walk. Consequently, the transition probability for the restricted type L between node i and j, in a weighted graph, is defined in Equation 4.7, where $e_{i,j}$ is the edge weight between the two nodes i and j. Given that we are removing the possibility of transitioning to any type other than the pre-defined one, the probabilities are normalized by the summation of edge weights that have that specific type as their end node. If such random walk is repeated for each node, the generated random walk corpus would reflect the separation of context needed to train the faceted models.

$$P_{i,j} = \begin{cases} \frac{f(e_{i,j})}{\sum_{k \in L, f(e_k) \in E_i} f(e_k)} & \text{if} \quad j \in L \\ 0 & \text{if} \quad j \notin L \end{cases} \tag{4.7}$$

Function f, in Equation 4.7, is the edge normalization function. Similar to our approach for entity embeddings, we use two normalization functions:

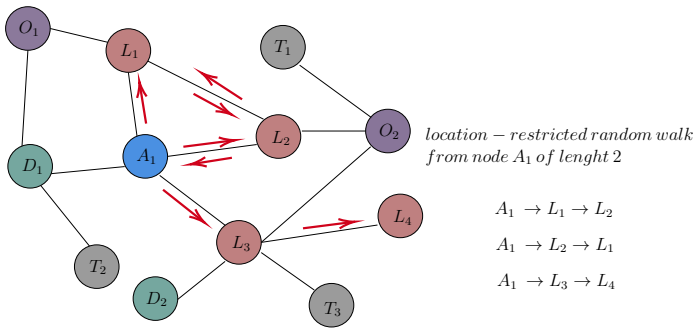


Figure 4.8: An example of type restricted random walk of length 2 for node types location and starting node A_1.

1. $f = \mathrm{id}$, identity function or no normalization. We denote these models by $f\mathrm{DW}_{id}$, where id shows the identity mapping.

2. $f = \log$, logarithmic normalization. These models are denoted by $f\mathrm{DW}_{log}$.

An example of the type-restricted random walk for type location is illustrated in Figure 4.8. Any nodes in the random walk have to be a location to be visited, otherwise, it is disregarded. Hence, by creating a type specific corpus with the type-restricted random walk, we can train a skip-gram model that learns the representation of all nodes with regards to a single entity type. By repeating this process for all node types and concatenating the results we achieve the faceted model.

4.5 Similarity Between Embeddings

In the previous chapter, we presented cosine similarity as the main similarity measure between word vectors. To take advantage of the separation in dimensions for faceted embedding, an additional vector multiplication for cosine similarity is defined, resulting into two types of similarity:

- *Full similarity:* Using the full dimensions of vectors for computing the cosine similarity or the dot product between two words. The full embedding vector contains the concatenation of entity types available in the data.

- *Partial similarity:* Using components based on query words. For example, if the similarity between an actor and a term is requested, only components related to actors and terms in both embeddings are kept to compute the cosine similarity. This method is used to test whether a particular dimensions keeps the most informative information about a type or not. Partial similarity can also be viewed as looking at the similarity between two word vectors in a chosen subspace. For example, looking only at cosine similarity between the organisation component of two embeddings will show how close they are in the organisation subspace.

4.6 Summary of Faceted Embeddings

In this chapter, we introduced three models for learning faceted embeddings with separable components, where each component defines the relation of a word to entities of a specific type. We modified the existing word embedding and graph embedding techniques to obtain a more flexible definition of context for a word, where for each component the context is limited to entities of a certain type. As input for all models, we used the co-occurrence graph extracted on annotated text, which contains the entity-entity relations and captures their co-occurrence with terms. From the word embedding methods, we modify GloVe and word2vec and from the graph-embedding models, we chose DeepWalk with type-restricted random walks. In Chapter 5, we evaluate the faceted models against the well-established word embedding methods on various term-based evaluation tasks and discuss their advantages and drawbacks.

5 Experimental Evaluation

In this chapter, we present the results for different methods of evaluation for entity embeddings and faceted models. In Section 5.1, most common methods for the evaluation of word embeddings are discussed and ones chosen for this study are presented. Section 5.2 explains the dataset used for training our models, along with the preprocessing steps and graph extraction. Evaluation setup and parameter tuning for all models is discussed in Section 5.3. Evaluation results for common tasks are shown in Section 5.4, where results for the entity embeddings and faceted models are discussed separately in Subsections 5.4.1 and 5.4.2 respectively. Finally, Section 5.5 contains the experimental exploration and visualization of embeddings neighbourhoods.

5.1 Evaluation of Word Embeddings

In general, the evaluation of word embeddings falls into two major categories: Extrinsic evaluation and intrinsic evaluation. In an extrinsic evaluation, trained embeddings are used as input of some downstream task and the performance is measured by a metric specific to that task. Examples of downstream tasks for embeddings are: POS tagging [Ling et al. 2015], named entity recognition [Al-Rfou et al. 2015], and dependency parsing [Chen and Manning 2014]. However, since evaluations are task-specific, an embedding that works well for one task may fail for another. The more popular evaluation technique for word embedding is intrinsic evaluation, which directly tests for syntactic or semantic relationships between words. These tasks analyze how well embeddings capture certain syntactic or semantic relations between a set of query and target words, referred to as *query inventory* [Schnabel et al. 2015]. Intrinsic tasks are easier and faster to test and provide a quick understanding of the system. The main drawback of intrinsic evaluation is that the notion of semantics is not universal [Bakarov 2018]. Two words are considered semantically similar if they are synonymous and can be substituted for each other in context. For instance, *"rose"* and *"flower"*. Semantic relatedness, however, is a broader concept. Words with any functional associations, like belonging to the same semantic field, are considered semantically related, e.g., *"rose"* and *"thorn"* [Kolb 2009]. It is not obvious if word embeddings should favor relatedness over similarity or vice versa. Furthermore, a high accuracy on these tasks does not necessarily results to good performance on real tasks unless some correlation between them is established. Thus, performing both intrinsic and extrinsic evaluation provides a better overview of the performance. Since there are no entity-centric extrinsic evaluation tasks in the literature, and few intrinsic datasets include entities, we focus on the performance in term-based intrinsic tasks. Moreover, we create some of our own datasets to incorporate the enti-

ties, but it is important to emphasize that the true potential of these entity-based models can only be investigated in the presence of suitable evaluation datasets. Following the approach by [Schnabel et al. 2015], we use three kinds of intrinsic evaluations. In the following, we describe each task and give a brief description of the datasets we use for evaluating each of them.

5.1.1 Relatedness

This task is based on datasets containing relatedness or similarity scores for pairs of words annotated by humans; the cosine similarity or Euclidean distance between the embeddings for two words should have a high correlation (Spearman or Pearson) with human similarity or relatedness scores. We choose seven datasets containing both relatedness and similarity scores.

1. *Similarity353:* 203 instances of similar word pairs from WordSim353 [Agirre et al. 2009]. Scores scale from 0 (totally unrelated words) to 10 (very related or identical words). Word pairs are classified as synonyms, antonyms, or identical, and unrelated pairs (pairs with average similarity less than or equal to 5).

2. *Relatedness353:* 252 instances of word pairs from WordSim353 [Agirre et al. 2009] that are not similar (are not present in the Similarity353 subset), but are still considered related by humans with an average similarity greater than 5, and unrelated pairs.

3. *MEN:* 3,000 word pairs with human-assigned similarity judgments, which instead of an absolute score is based on comparative judgments on two pair exemplars at a time. Each person was given a binary choice of *"right"* or *"wrong"* and in total each pair was compared 50 times, thus, obtaining a final score on a 50 point scale [Bruni et al. 2014].

4. *RG65:* 65 pairs with annotated similarity scaling from 0 to 4, where the similarity between two words is measured by the similarity of their meaning and the amount of overlap for each definition of context [Rubenstein and Goodenough 1965].

5. *RareWord:* 2,034 rare word pairs with human similarity scores. For creating the dataset a list of rare words is selected and for each another word is found, which is not necessarily rare. The human judement is assesed on how similar each pair is on a scale of 0 to 10 [Luong et al. 2013].

6. *SimLex-999:* 999 pairs of human-labeled examples of semantic relatedness. The dataset contains a selection of adjective, verb, and noun concept pairs. The human judgment scores range from 0 to 10 [Hill et al. 2015].

7. *MTurk:* 771 word pairs with semantic relatedness scores from 0 to 5, where scores were collected on Amazon Mechanical Turk [Radinsky et al. 2011].

Similarity between the word pairs is typically computed by the cosine similarity of their respective word vectors. In the case of faceted models, we can investigate the similarity between two words further by looking at separate components. For this purpose, we take advantage of the partial similarity

defined in Chapter 4 and compute the correlation between human annotations and the partial similarity of each component, separately. Furthermore, assuming that most of the information about the relationship between the two words is in the component corresponding to their types, we compute the similarity between types in the relation. For example, if two words are of types actor and term, the actor and term components are kept and the cosine similarity is computed based on the two components related to types of words in question.

5.1.2 Analogy

Table 5.1: Types of questions in the Google Analogy test, in total nine morphological and five semantic questions are present in the dataset. From the semantic question, capital-common-countries, capital-world, and city-in-state are location entities.

Morphological	
gram1-adjective-to-adverb:	an adverb with its adjective.
gram2-opposite:	noun and its opposite.
gram3-comparative:	an adjective with its comparative.
gram4-superlative:	an adjective with its superlative.
gram5-present-participle:	participles and their verb in infinitive.
gram6-nationality-adjective:	country with its nationality.
gram7-past-tense:	verbs in infinitive and past form.
gram8-plural:	nouns in single and plural from.
gram9-plural-verbs:	verbs in singular and plural.
Semantic	
capital-common-countries:	selection of countries and their capitals.
capital-world:	all countries and their capitals.
family:	family relations such as brother and sister
city-in-state:	states and their cities name (US).
currency:	name of a country with its currency.

This task was popularized by the word2vec paper [Mikolov et al. 2013a], where the goal is to find a term x for a given term y, such that $x : y$ best resembles a sample relationship $a : b$. In other words, given the triple (a, b, x), the aim is to find the target word y correctly. To find the target word we do the following:

- Compute $w_a - w_b = w_{a-b}$, where w_a and w_b are the embeddings for word a and b.
- Compute the vector for the prediction $w_{\hat{y}} = w_{a-b} + w_x$. The idea is that the direction of the relation should remain the same.
- Find the nearest neighbour to $w_{\hat{y}}$ (the one with the highest cosine similarity), if it is w_y, the analogy was found correctly.

For entity embeddings, where the type of the embedding is known, we can also consider an easier, type-specific variation of this task, which only considers neighbours that match a given entity class. In the type-specific variation we only consider candidates that have the same entity type as the words in the triple (a, b, x), thus making the task easier. We use two dataset for evaluating this task:

1. *GA*: The *Google Analogy* data consists of $19,544$ pairs ($8,869$ semantic and $10,675$ syntactic), with 14 types of relations (9 morphological and 5 semantic), used in the word2vec paper [Mikolov et al. 2013a]. Besides terms, it contains some location entities of cities and countries. Types and their meaning are shown in Table 5.1.

2. *MSR:* The *Microsoft Research Syntactic analogies* dataset contains $8,000$ morphological questions. The questions contain base/comparative/superlative forms of adjectives; singular/plural forms of common nouns; possessive/non-possessive forms of common nouns; and base, past and 3rd person present tense forms of verbs [Mikolov et al. 2013b]. All word are terms.

5.1.3 Categorization

When projecting the word embeddings in a 2 or 3-dimensional space, we expect that similar words form meaningful clusters. Embeddings can be projected using *t-SNE* [Maaten and Hinton 2008] or PCA [Shlens 2014]. For categorization task, vectors of all words are either visualised for exploratory analysis or clustered using a clustering algorithm of choice and the purity of the returned clusters is computed as a metric for this task (other metrics for clustering can also be used, but purity is the most common) [Manning et al. 2008]. For evaluating embeddings on categorization task, we use two datasets from the Lexical Semantics Workshop, which do not contain named entities. Additionally, we created three test sets using *Wikidata Query Service* to find entities of type actors, location, and organization. In each case, up to 30 entities form each category were chosen based on the highest frequency in our training data.

1. *ESSLLI_1a:* Acronym for the European Summer School in Logic, Language and Information. The goal of the task is to group concrete nouns into semantic categories and contains 44 concrete nouns that belong to six semantic categories [Baroni et al. 2008].

2. *ESSLLI_2c:* The goal of the sub-task is to group verbs into semantic categories and contains 45 verbs that belong to nine semantic classes [Baroni et al. 2008].

3. *Cities:* 150 major cities in the U.S., the U.K., and Germany.

4. *Politicians:* 150 politicians from the U.S., the U.K., and Germany.

5. *Companies:* 110 software companies, Web services, and car manufacturers.

5.2 Training Data

For training, we use 209, 023 news articles from English-speaking news outlets, collected from June to November 2016 [Spitz and Gertz 2018]. The data contains a total of 5, 427, 383 sentences and 4, 468, 993 tokens. To train regular word embeddings, we use the raw article text, with only common preprocessing steps. Pre-processing steps contain tokenization of all articles into words and removal of stop-words and punctuations. Non-alphabetic characters, like Chinese or Arabic vocabulary, are disregarded and multiple white spaces are reduced to only one. Any HTML tags or numeric values are also removed from the text, to produce a clean document, containing only terms relevant to the model. Since all the numeric values are removed, any date instance that is presented as numerical expression is also removed. Additionally, all words are converted to their lower-case form to avoid multiple representations of a single term.
For annotated embeddings, prior to named entity annotation, POS tagging is performed by the Stanford POS tagger [Toutanova et al. 2003]. We removed punctuation and stop-words and filtered terms in the text to certain POS-tags. Unnecessary tags are removed, namely wh-determiner, wh-pronouns, wh-adverbs, common verbs, such as *"have"* and *"do"* in past or present, interjections (e.g. *"ah!"*, *"dear me!"*) and other terms that mostly fall into the stop-words category such as: predeterminer (*"both"* and a *"lot of"*), possessive endings, prepositions (*"until"*, *"before"*,...), determiner (*"a"*, *"the"*, *"every"*) and coordinating conjunction (*"and"*, *"but"*, *"or"*). Afterwards, we extract named entities with *Ambiverse*[1], a state-of-the-art commercial annotator that links entity mentions to Wikidata identifiers and selects named entities that correspond to persons, locations, and organizations. For temporal expressions (date entities) we use *HeidelTime* [Strötgen and Gertz 2013]. HeidelTime is a multilingual temporal tagger that extracts temporal expressions from documents and normalizes to a standard format. Hence, all dates, regardless of their style, are normalized.
To generate input for the graph-based embeddings, we extract an implicit graph of locations, organizations, persons (actors), dates, and terms, using the extraction code of the LOAD model [Spitz and Gertz 2016]. In the LOAD model, we include term co-occurrences only inside sentences and entity-entity co-occurrences up to a default window size of five sentences, thus, there exists an edge between terms and other words only if they appear in the same sentence. The generated graph contains $V = 93,390$ nodes and $E = 9,584,191$ edges. From which $D = 2,442$ are date entities, $L = 4,659$ location, $A = 10,537$ actors, $O = 3,789$ organisations and $T = 71,963$ are terms. The majority of the graph consists of terms and the type date has the fewest nodes in the graph. Evaluation datasets contain words that are not present in the training vocabulary, for this reason, each dataset is filtered accordingly.

[1] https://www.ambiverse.com

5.3 Parameter Tuning

We perform extensive parameter tuning for each model. In this section, we report the best parameter setting for each model. To make the embeddings comparable, all embeddings are trained with 100 dimensions. In the following, we first discuss the best parameter settings for annotated and raw models and then for faceted embeddings. Because of the random initialization at the beginning of the training, all entity and raw models are trained 10 times and the average performance is reported. However, since faceted models are only an extension to annotated models and are only trained for experimental analysis we trained a single model for each. For all models, instead of a typical grid search, we chose a coarse-to-fine method, where instead of checking all possible combinations, we randomly sampled some parameters and focused on areas, where better models are trained.

5.3.1 Parameter settings for entity embeddings and word embeddings

Word2vec-based models namely, the word2vec on the raw data (rW2V) and the word2vec on the annotated corpus (aW2V), are trained with a learning rate of 0.015 and a window size of 10. We use 8 negative samples on the raw data, and 16 on the annotated data. In addition to data pre-processing, we removed words with a frequency of less than 3, as there is not enough data to learn a meaningful representation of them.

GloVe-based models, namely the GloVe model on the raw data (rGLV) and on the annotated text (aGLV) are trained with a learning rate of 0.06 and a window size of 10. For the weighting function, we did not use value suggested by authors, instead performed additional parameter tuning, which resulted in a scaling factor of 0.5 and a maximum cut-off of 1000 to obtain best performance. Words that occur less than 5 times are removed from the input.

DeepWalk models use 100 random walks of length 4 from each node, for both identity (DW$_{id}$) and logarithmic (DW$_{log}$) mappings. We did not experiment with walk lengths larger than 5, because the co-occurrence graph is dense and has a relatively small diameter. Thus, longer walks would introduce unrelated words into contexts and reduce the performance. For the skip-gram model, which was trained on the random walk corpus, we use a learning rate of 0.015 and 64 negative samples with the window-size of 10. We removed words that were visited less than 3 times in the random walk corpus.

VERSE models have relatively few parameters for tuning. We use a learning rate of 0.025 and 16 negative samples.

Because of different inputs for each embedding model, comparison between them is not trivial. One main challenge is the fact that the training process of graph-based and textual methods is essentially incomparable. Graph-based models do not have a textual corpus and require an additional step of

creating something similar either with a random walk or node sampling. Typically, the performance of neural network based models increases with the number of iterations and more data. While textual models consider one pass through the corpus as one iteration, an iteration for a graph-based model is a different concept and not the only factor for increasing performance. The data size in graph-based models is not limited to the corpus but the sampling process. Number of random walks that is used in DeepWalk corresponds directly to the performance, more random walks results in a larger corpus and, consequently, better performance of the skip-gram model trained on it. Therefore, even if the number of iterations for the skip-gram model is fixed, more random walks can increase performance. On the other hand, increasing the number of random walks also increases the runtime, and thus, a trade-off between the quality of the model and speed has to be found. In contrast, the VERSE model has no notion of iteration and samples nodes for positive observations from the empirical similarity distribution and negative observations from a noise distribution. The larger the sample size, the better the performance, and the onger the runtime. Hence, to approach a fair evaluation, we use similar training times for all models (roughly 10 hours per model on a 100 core machine). We fix the number of iterations of the textual models and DeepWalk's skip-gram at 100. For VERSE, we use $50,000$ sampling steps to obtain a comparable runtime.

5.3.2 Parameter settings for faceted models

Faceted GloVe model is trained with a learning rate of 0.05. For the best performance, we use focal addition after training to combine focal and context embeddings. The scaling factor for the weighting function is set to 0.75 and maximum cut-off of 1000 is used to remove edge weight below the threshold. All nodes with less than 5 edges are removed from the training data before generating the weighted adjacency matrix. Moreover, we only report results for $f\text{GLV}_{sep}$, which refers to the separate cost function as the performance of both cost functions are similar. To achieve the faceted components, each component is trained separately with the same parameter setting and embedding size of 20, which after concatenation creates the full 100 dimensional embedding.

The Faceted word2vec model is trained with the learning rate of 0.015 and 32 negative sample. To achieve the embedding size of 100, each component has an embedding size of 20. Any words that have less than 3 edges in the co-occurrence graph are removed.

Faceted DeepWalk models use 100 random walks of length 4 from each node. We train the model once using logarithmic normalization ($f\text{DW}_{log}$) and once without any normalization ($f\text{DW}_{id}$). The skip-gram model uses a learning rate of 0.015 and 32 negative samples. The window size is set to 10 and any node that is visited less than 3 times in the random walk corpus is removed.

Table 5.2: Word similarity results. Shown are the Pearson correlations between the cosine similarity of the embeddings and the human score on the word similarity datasets. The two best values per task are highlighted.

	rW2V	rGLV	aW2V	aGLV	DW$_{id}$	DW$_{log}$	VRS
Similarity353	**0.700**	0.497	**0.697**	0.450	0.571	0.572	0.641
Relatedness353	**0.509**	0.430	0.507	0.428	0.502	0.506	**0.608**
MEN	**0.619**	0.471	**0.619**	0.469	0.539	0.546	**0.640**
RG65	**0.477**	0.399	0.476	0.386	0.312	0.344	**0.484**
RareWord	**0.409**	0.276	**0.409**	0.274	**0.279**	0.276	0.205
SimLex-999	**0.319**	0.211	**0.319**	0.211	**0.279**	0.201	0.236
MTurk	**0.647**	0.493	0.644	0.502	0.592	0.591	**0.687**
average	**0.526**	0.400	**0.524**	0.389	0.439	0.433	0.500

5.4 Evaluation Results

In the following, we discuss the results for relatedness, analogy and clustering tasks for entity embeddings and faceted models, separately. We compare our approaches against classical word embeddings trained on raw text and analyse their weaknesses and strength. For faceted models, we take advantage of the separable dimensions and in addition to testing embeddings in their full dimension, we also look at each component separately. Results for the faceted GloVe model is not presented on all tasks, because of the poor performance.

5.4.1 Evaluation of entity embeddings

We evaluated our models on all three common intrinsic evaluation tasks. For each task, the values reported in tables are the average score of 10 models.

Results of the relatedness task for entity embeddings are shown in Table 5.2. On this data, word2vec model predominantly performs better than GloVe on both raw and annotated text. Nevertheless, the performance of both models degrades slightly after annotation. Therefore, to create embeddings of entities, naively using a word embedding method on an annotated corpus results in a loss in performance. However, since all datasets consist mostly of terms, this observation cannot be extended to entity-centric evaluations. DeepWalk-based models perform better than GloVe, but do poorly overall. The logarithmic normalization boosts the performance of DeepWalk-based models slightly but not enough to compete with word2vec. Results for the VERSE model are rather diverse, in a sense that it works very well on some tasks, but is worse than word2vec trained on the raw data for rare words and the SimLex dataset. This is likely caused by the conceptual structure of the co-occurrence graph on which VERSE is trained on, which captures relatedness and not similarity as tested by SimLex. In contrast, on datasets, such as Relatedness353 and MEN, which are designed for relatedness tasks, VERSE is clearly the better choice. On average, word2vec-based models outperform all other

Table 5.3: Word analogy results. Shown is the prediction accuracy for the normal analogy tasks and the variation where predictions are limited to the correct entity type. The best two values per task and variation are highlighted.

	normal analogy							typed analogy				
	rW2V	rGLV	aW2V	aGLV	DW$_{id}$	DW$_{log}$	VRS	aW2V	aGLV	DW$_{id}$	DW$_{log}$	VRS
GA	0.013	**0.019**	0.003	0.015	0.009	0.009	**0.035**	0.003	**0.016**	0.110	0.110	**0.047**
MSR	**0.014**	**0.019**	0.001	**0.014**	0.002	0.002	0.012	0.001	**0.014**	0.002	0.002	**0.012**
avg	0.013	**0.019**	0.002	0.014	0.005	0.005	**0.023**	0.002	**0.015**	0.006	0.006	**0.030**

models on similarity tasks, both the model trained on raw text and on the annotated data have the highest scores. Thus, we conclude that for purely term-based tasks, word2vec is the best choice for measuring word similarity, while VERSE captures the relatedness better.

Table 5.3 shows the accuracy achieved by entity-based and term-based models in the words analogy task. Overall, results are quite poor for all models, which we attribute to the size of the training data. Successful models on this tasks are usually trained on billions of tokens, which is much larger than the size of the corpus used in this thesis. GloVe performs better than word2vec for this task on both raw and annotated data, while VERSE has a slight edge. When using the typed search for the analogy task, in which we only look at candidates that share the same type as entities in question, the task becomes easier and results improve. The improvement is more noticeable for entity-centric questions, contained in the GA task. If we consider only the subset of $6,892$ location targets for the GA task, we find that graph-based models perform much better, with VERSE being able to predict up to $1,662$ (24.1%) of location targets on its best run, while aW2V and aGLV are only able to predict 14 (0.20%) and 16 (0.23%), respectively. This shows the potential advantage of entity-based methods for analogy tasks that contain them. Nevertheless, for the MSR task, which does not contain named entities, applying the typed search does not improve results.

The purity of clusters created with agglomerative clustering and mini-batch k-means for the categorization tasks are shown in Table 5.4. Since the Cities, Politicians and Companies contain multi-word entities, e.g., politician names like *"Barack Obama"*, they do not directly exist in raw embedding models. To represent such words, we take the average of vectors of individual words in the entity's name and use the average vector for clustering tasks, e.g., average of vectors *"barack"* and *"obama"*. Although raw models do not embed the named entities directly, rW2V and rGLV create clusters with the best purity even on entity-based datasets. Nonetheless, most purity values lie in the range from 0.45 to 0.65 and no method is exceptionally bad. It is worth noting that the clustering algorithm also plays a part in the obtained results. K-means clustering has a randomized nature and the results differ based on the initial conditions. However, on average, the k-means algorithm forms clusters with higher purity when used on rW2V, while agglomerative clustering produces better clusters on rGLV. DeepWalk-based models also tend to have a higher purity using agglomerative clustering. Hence, it is the combination of clustering algorithm and the underlying embedding used

Table 5.4: Categorization results. Shown is the purity of clusters obtained with k-means and agglomerative clustering (AC). The best two values are highlighted. For the raw text models, multi-word entity names are the mean of word vectors.

		rW2V	rGLV	aW2V	aGLV	DW$_{id}$	DW$_{log}$	VRS
k-means	ESSLLI_1a	**0.575**	0.545	**0.593**	0.454	0.570	0.520	0.534
	ESSLLI_2c	0.455	0.462	**0.522**	0.464	0.471	0.480	0.584
	Cities	**0.638**	**0.576**	0.467	0.491	0.560	0.549	0.468
	Politicians	**0.635**	0.509	0.402	0.482	0.470	0.439	**0.540**
	Companies	**0.697**	**0.566**	0.505	0.487	0.504	0.534	0.540
	average	**0.600**	0.532	0.498	0.476	0.515	0.504	**0.533**
AC	ESSLLI_1a	0.493	**0.518**	0.493	0.440	0.486	0.502	**0.584**
	ESSLLI_2c	0.455	0.398	0.382	0.349	**0.560**	**0.408**	0.442
	Cities	0.447	**0.580**	0.440	0.515	0.364	**0.549**	0.359
	Politicians	0.477	**0.510**	**0.482**	0.480	0.355	0.360	0.355
	Companies	**0.511**	**0.519**	0.475	0.504	0.474	0.469	0.473
	average	**0.477**	**0.505**	0.454	0.458	0.448	0.458	0.443

that determines the performance on categorization. Moreover, this tasks only focuses on the terms and entities provided by the datasets,thus, results do not give us insight into the spatial mixing of terms and entities. We consider this property in our visual exploration in Section 5.5, where we visualize all models using t-SNE.

In summary, for term-based intrinsic evaluation tasks, embedding entities by state-of-the-art methods on annotated corpus has an acceptable performance, yet degrades in comparison to embeddings on the raw corpus, and thus, is not the best option. This lack in performance can be restored to some degree by using graph-based techniques on a co-occurrence graph extracted from the annotated text. In particular for a tasks that include entities or require a measure of relatedness, the VERSE model often has better performance, even compared to embeddings on raw text. For clustering and word similarity tasks, however, using the normal word embeddings tends to achieve better results. In general, when using entity-based models attributes of the downstream task has to be taken into account, as different embedding techniques demonstrate different properties.
Although term-based evaluations shine some light into the performance of entity-based embeddings, the true potential of such models is still unknown and more entity-specific tasks are needed for assessing the performance on entity-centric tasks . Thus, in Section 5.5, we explore the usefulness of different embeddings for entity-centric tasks.

Table 5.5: Word similarity results for faceted models for the normal word similarity and similarity between the components related to word types. Shown are the Pearson correlations between the cosine similarity of the embeddings and the human score on the word similarity datasets. The two best values per task are highlighted.

	normal similarity					related similarity		
	rW2V	rGLV	fW2V	fDW$_{id}$	fDW$_{log}$	fW2V	fDW$_{id}$	fDW$_{log}$
Similarity353	**0.700**	**0.497**	0.394	0.068	0.070	0.394	0.336	0.313
Relatedness353	**0.509**	**0.430**	0.324	0.165	0.167	0.323	0.250	0.244
MEN	**0.619**	**0.471**	0.372	0.197	0.195	0.372	0.334	0.326
RG65	**0.477**	**0.399**	0.153	-0.089	-0.035	0.153	0.096	0.096
RareWord	**0.409**	**0.276**	0.094	0.048	0.048	0.094	0.212	0.212
SimLex-999	**0.319**	**0.211**	0.135	0.062	0 .050	0.135	0.101	0.104
MTurk	**0.647**	**0.493**	0.448	0.137	0.146	0.448	0.347	0.364
average	**0.526**	**0.400**	0.269	0.084	0.0916	0.268	0.239	0.237

5.4.2 Evaluation of faceted embeddings

Since a comparable faceted model does not exist in the literature, we compare our models against regular word embeddings trained on the raw text and analyse whether the separation of components is harmful to the performance of embeddings on common tasks.

The result for the relatedness task on faceted models is shown in Table 5.5. The normal similarity is computed based on the cosine similarity, where all the components of faceted embeddings (actor, location, organisation, date, and term) are used to compute the cosine similarity. Furthermore, we introduce a notion of related similarity, where only components related to types of embeddings in question are used (e.g., if both words are terms only the term component is used). Since majority of words are terms, the related similarity is closely correlated with the partial similarity of only the term component. The result for the fGLV$_{sep}$ is not shown in the table, because of extremely poor results. fGLV$_{sep}$ achieved the score of -0.021 on the MEN dataset and the correlation 0.122 was its best performance on similarity tasks. Therefore, it showed almost no correlation with human judgement scores and is disregarded. Faceted word2vec and faceted DeepWalk capture similarity and relatedness among words better than the faceted GloVe model. Nonetheless, classical word embedding methods produce better results for all datasets and are undoubtedly the better choice for this task. Using only related components, enhances the results, in particular for the DeepWalk model, which indicates that some components of the faceted model do not contain useful information about the relationships among words and might even have a negative effect on the result. However, this is not the case for fW2V, which not only has the highest scores for both normal and related similarity tasks, also the score remains stable when removing unrelated components.

To investigate the impact of each component separately on the relatedness task, we compute the

Table 5.6: Word similarity results for faceted models for the partial similarity between different components of the faceted models. Shown are the Pearson correlations between the cosine similarity of each component and the human score on the word similarity datasets.

	Term similarity			Actor similarity			Location similarity			Organisation similarity			Date similarity		
	fW2V	fDW$_{id}$	fDW$_{log}$	fW2V	fDW$_{id}$	fDW$_{log}$	fW2V	fDW$_{id}$	fDW$_{log}$	fW2V	fDW$_{id}$	fDW$_{log}$	fW2V	fDW$_{id}$	fDW$_{log}$
Similarity353	0.397	0.338	0.315	-0.065	-0.096	-0.045	0.076	-0.135	-0.105	-0.065	-0.033	-0.056	0	-0.143	-0.098
Relatedness353	0.333	0.254	0.247	0.098	0.123	0.005	0.098	-0.034	0.054	0.098	0.089	0.067	0.137	0.038	0.103
MEN	0.373	0.327	0.334	0	0.002	-0.004	0	0.058	0.055	0	-0.007	0.002	0	0.025	0.016
RG65	0.153	0.096	0.096	0	0.009	-0.025	0	-0.126	-0.163	0	0.049	0.009	0	-0.135	-0.089
RareWord	0.094	0.212	0.212	0	0.022	0.025	0	0.002	-0.004	0	-0.021	-0.024	0	-0.031	-0.026
SimLex-999	0.135	0.101	0.101	0	0.006	-0.026	0	-0.032	-0.012	0	-0.001	-0.006	0	0.009	-0.037
MTurk	0.448	0.347	0.347	0	-0.054	-0.031	0	0.007	-0.007	0	0.038	0.033	0	-0.076	-0.030
average	0.271	0.254	0.250	0.022	0.001	-0.014	0.022	-0.037	-0.026	0.022	0.016	0.004	0.137	-0.045	-0.023

partial similarity with respect to each entity type separately. The result for the partial similarity is shown in Table 5.6. As expected, the term component has the highest correlation with the human judgement for all datasets. Because of the way fW2V is constructed, we observe many zeros for types other than term. This is due to the construction of components for the fW2V, where the components are set to a small number if the word does not have a relation with any entity corresponding to the component type. Setting all values of a component to a small constant number if the word does not have any entity of that type in its context, has some implications. If majority of word pais in the dataset have a constant components, majority of them would have cosine similarity of one. Consequently, If most of the cosine similarities are one for the words in the dataset, their correlation with human judgement is negligible. In the same manner, for faceted DeepWalk models, correlation values are also a small number in the vicinity of zero for all non-term components. But the effect is not as extreme as fW2V, which is due to the random initialization of all components at the beginning of training. For some unrelated components, the correlation values are even negative, which explains the reason behind the extreme difference between the score of related and normal similarity for DeepWalk-based methods in Table 5.8, as the negative correlation of some component reduces the overall score of the model.

In general, separation of different types has a negative effect on similarity tasks that are purely term-based. Nonetheless, computing the related similarity or using only the term component, can enhance the performance. In any case, word embedding methods trained on the raw text are clearly the better option. By looking at different components for term-based datasets, we can deduce that most of the information about the similarity and relatedness of two terms lies in the surrounding terms and looking at only entities can be detrimental to this task. If datasets with specific entity types are provided, more experiments can be done to test whether the components of a specific-type carries the most information about the words semantic or not. We leave the creation and evaluation of such test cases as future work.

Table 5.7: Word analogy results for faceted embedding. Shown is the prediction accuracy for the normal analogy tasks and the variation where predictions are limited to the correct entity type. The best value per component for each task it highlighted.

	normal analogy					typed analogy		
	rW2V	rGLV	fW2V	fDW$_{id}$	fDW$_{log}$	fW2V	fDW$_{id}$	fDW$_{log}$
GA	0.013	**0.019**	**0.057**	0.001	0.002	**0.058**	0.001	**0.002**
MSR	**0.014**	**0.019**	0.003	0	0	0.003	0	0
average	0.013	**0.019**	**0.030**	0	0.001	**0.030**	0	**0.001**

Table 5.8: Categorization results for faceted embeddings and the word embedding on raw text. Shown is the purity of clusters obtained with k-means and agglomerative clustering (AC). The best two values are highlighted. For the raw text models, multi-word entity names are the mean of word vectors.

		rW2V	rGLV	fW2V	f DW$_{id}$	fDW$_{log}$
k-means	ESSLLI_1a	**0.575**	**0.545**	0.432	0.523	0.523
	ESSLLI_2c	0.455	0.462	0.400	0.444	0.422
	Cities	**0.638**	0.576	0.550	0.567	**0.633**
	Politicians	**0.635**	0.509	0.500	**0.513**	0.493
	Companies	**0.697**	0.566	0.482	**0.600**	0.527
	average	**0.600**	0.532	0.473	**0.529**	0.520
AC	ESSLLI_1a	**0.493**	**0.518**	0.455	0.432	0.432
	ESSLLI_2c	**0.455**	0.398	0.378	**0.444**	**0.444**
	Cities	0.447	**0.580**	0.470	0.347	0.347
	Politicians	**0.477**	**0.510**	0.426	0.340	0.353
	Companies	**0.511**	**0.519**	0.463	0.464	0.464
	average	**0.477**	**0.505**	0.432	0.405	0.408

In Table 5.7, the accuracy achieved by the analogy task for faceted models and normal word embeddings is shown. The fGLV$_{sep}$ was unable to retrieve any of the analogies correctly and is removed from the results. The faceted DeepWalk models also achieve very poor performance for analogy tasks. Nonetheless, fW2V manages to outperform the normal word embeddings for the Google Analogy task but its performance degrades for the purely term-based dataset of MSR. The boost in performance comes from the location entities in the GA dataset. From the subset of $6,892$ location targets for the GA task, we find that fW2V is able to predict 963 (14%) of the locations targets, while aW2V and aGLV are only able to predict 14 (0.20%) and 16 (0.23%), respectively. We observed the same effect for the entity embeddings evaluation in Section 5.4.1, where the models on the annotated text are more successful in predicting entity-based relations. Among the models on annotated text, the graph embedding VRS is able to predict $1,662$ (24.1%) and outperforms the

Table 5.9: Categorization results for faceted models. Shown is the purity of clusters obtained with k-means (KM) and agglomerative clustering (AC). For the raw text models, multi-word entity names are the mean of word vectors.

		Term component			Actor component			Location component			Organisation component			Date component		
		*f*W2V	$f\text{DW}_{id}$	$f\text{DW}_{log}$	*f*W2V	$f\text{DW}_{id}$	$f\text{DW}_{log}$	*f*W2V	$f\text{DW}_{id}$	$f\text{DW}_{log}$	*f*W2V	$f\text{DW}_{id}$	$f\text{DW}_{log}$	*f*W2V	$f\text{DW}_{id}$	$f\text{DW}_{log}$
KM	ESSLLI_1a	0.500	0.545	0.568	0.295	0.386	0.386	0.295	0.432	0.455	0.295	0.409	0.477	0.295	0.454	0.386
	ESSLLI_2c	0.444	0.422	0.444	0.111	0.311	0.333	0.111	0.356	0.356	0.111	0.356	0.377	0.111	0.311	0.355
	Cities	0.570	0.467	0.506	0.157	0.500	0.473	0.490	0.560	0.500	0.510	0.507	0.520	0.342	0.380	0.413
	Politicians	0.432	0.453	0.486	0.493	0.527	0.500	0.338	0.427	0.440	0.460	0.440	0.460	0.338	0.367	0.373
	Companies	0.518	0.481	0.481	0.455	0.481	0.500	0.455	0.500	0.500	0.455	0.527	0.564	0.455	0.464	0.464
	average	0.493	0.474	0.497	0.302	0.441	0.438	0.338	0.455	0.450	0.366	0.448	0.480	0.308	0.395	0.398
AC	ESSLLI_1a	0.454	0.522	0.500	0.409	0.363	0.341	0.409	0.341	0.341	0.409	0.341	0.341	0.409	0.340	0.364
	ESSLLI_2c	0.422	0.511	0.467	0.288	0.311	0.333	0.288	0.311	0.333	0.288	0.356	0.288	0.288	0.311	0.333
	Cities	0.570	0.340	0.346	0.467	0.347	0.380	0.403	0.567	0.573	0.503	0.373	0.367	0.349	0.367	0.367
	Politicians	0.432	0.353	0.346	0.351	0.507	0.566	0.351	0.347	0.367	0.405	0.353	0.353	0.351	0.353	0.347
	Companies	0.463	0.464	0.490	0.464	0.473	0.455	0.455	0.482	0.482	0.518	0.482	0.473	0.455	0.455	0.464
	average	0.468	0.438	0.430	0.396	0.400	0.415	0.381	0.410	0.419	0.425	0.381	0.364	0.370	0.365	0.375

*f*W2V. Since the input of *f*W2V is the edge of the co-occurrence graph extracted from an annotated text, it is expected to have similar performance to entity-based embeddings. Unlike the entity-based embedding, however, the typed search, in which we only look at candidates of the same type as the entities in question, does not have a huge impact on the results.

The purity of clustering achieved by k-means and agglomerative clustering methods, for faceted models and the word embeddings on the raw text, are shown in Table 5.8. For faceted models, concatenation all components is used to generate each embedding. Multi-words in city, politician and company names, for raw embedding models, are presented by the average of embeddings for the individual words in their names. Predominantly, word embedding on the raw text create the best clusters, even for entity-based datasets. Despite the fact that *f*W2V had the best performance among faceted models in other evaluation tasks, the model is not able to cluster similar words effectively and faceted DeepWalk models tend to have better performance. Similar to entity-based embeddings, we observe the effect of clustering algorithm on the quality of clusters, where DeepWalk based models perform better using mini-batch k-means in comparison with agglomerative clustering.
To analyse the effect of different components in faceted models, we perform the categorization task using each component separately. The purity of the clustering for k-means and agglomerative clustering is shown in Table 5.9. Once again, the component corresponding to terms achieves the highest score for all models, in particular for term-based datasets. This effect is most noticeable in the case of *f*W2V, where for the rest of the components the purity is mostly a constant number, which is not surprising, since majority of the words in test sets are terms. For example, in case of ESSLLI_1a and ESSLLI_2c datasets, for k-means, the result for actor, location, organisation, and date are 0.295 and 0.111, respectively, corresponding to the lowest possible purity score when all the words are mapped to the same point. Values are more diverse for Faceted DeepWalk models since all components have

non-constant values. In the case of Faceted DeepWalk model, for datasets that contain named entities, components that correspond to the type of the entities in question become significant. The actor component achieves the highest purity score on the dataset with politicians, while the location and organisation tend to separate companies of different countries better.

In summary, faceted models in their complete form (all the components concatenated) are not able to compete with word embeddings on term-based intrinsic tasks. Using components that are relevant for types of entities in question, restores some of the lack of performance. Furthermore, analysing components separately shows that terms that appear in the context of a word tend to be the most influential in its meaning and define its relation to other words. Without terms, named entities on their own are not enough to generate a useful representation of words. In Section 5.5, we explore the different components and their significance further using entity-centric tasks.

5.5 Experimental Exploration

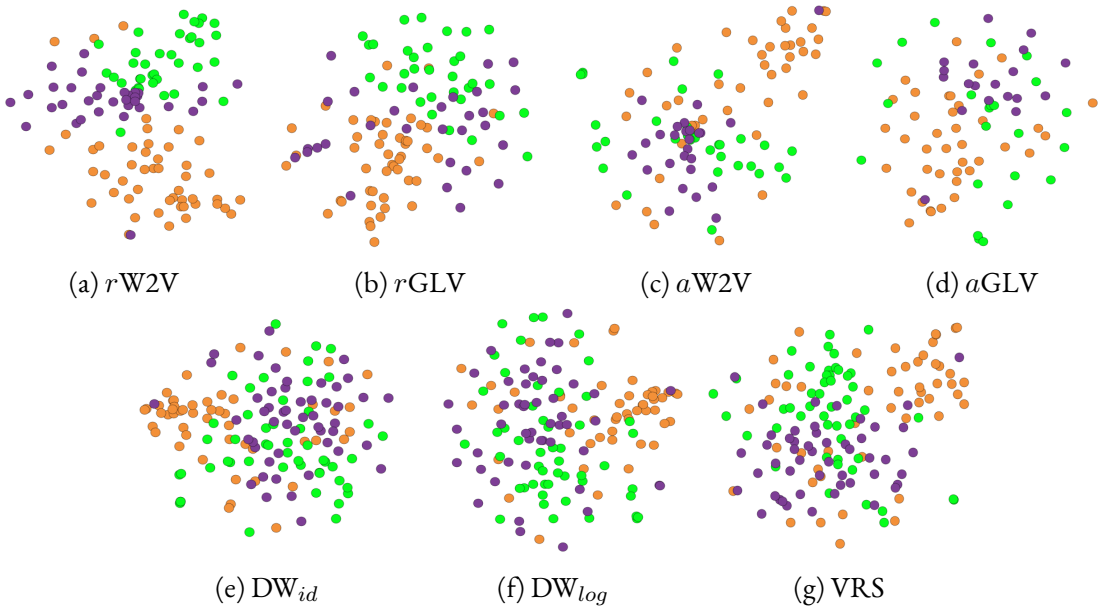
Entity embeddings and faceted models lack enough entity-centric tasks, while their evaluation depends mainly on their performance on such tasks. For a clear analysis of their potential use-cases, entity-based models require datasets that assess entity-entity relations. For this reason, we explore their utility for tasks invoking named entities through visualization and experimental evaluation of selected test cases. In Subsection 5.5.1, we look at possible benefits of entity embeddings through clustering of similar named entities and investigating the entity neighbourhoods. In Subsection 5.5.2, we analyse individual components of faceted models for clustering and explore the difference in neighbourhood structure based on the component type.

5.5.1 Analysis of entity embeddings

To investigate possible use-case and benefits of entity embeddings, we have to look at their performance on downstream tasks. Nonetheless, such tasks are not yet available for embeddings of entities and we can not asses the performance of our models using only term-based methods. Therefore, we visualize the one dataset from categorization task and analyse the neighbourhood for entities to obtain an impression of possible benefits that these embeddings can offer.

5.5.1.1 Entity clustering

One way to investigate how embeddings on annotated text enhance the representation of entities is to observe whether they are able to cluster similar entities in the embedding space. Since entity-based models are trained on data, where the named entities are annotated and disambiguated, we expect that such models create more meaningful clusters in comparison to traditional word embedding techniques. For this purpose, we project embeddings from the Cities dataset into a 2-dimensional space using t-SNE. Since the training data is taken from news, we expect cities within a country to be



(a) rW2V (b) rGLV (c) aW2V (d) aGLV

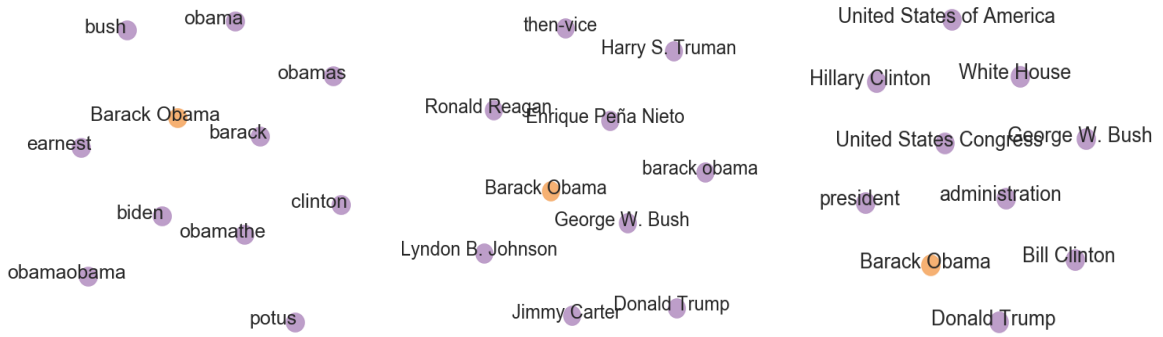
(e) DW$_{id}$ (f) DW$_{log}$ (g) VRS

Figure 5.1: t-SNE projections of the embeddings for U.S. (purple), British (orange), and German (green) cities. For the raw text models, multi-word entity names are represented as the mean of word vectors.

spatially correlated. Projections for all models is shown in Figure 5.1, where cities in U.S. is marked in purple, British cities in orange, and German cities in green. Since multi-word city names do not directly exist in models on raw text, we present them by the average of embeddings of their individual word. For most clustering tasks entity embeddings have poor performance and no obvious pattern can be found in their projections. Although aW2V and VRS tend to create small batches of cities that belong to the same country, they do not show an obvious distinction between them. Word embeddings on raw text perform much better for this task. Nevertheless, without the knowledge of such entity names for multi-word entities, raw models are unable to recognize them as a distinct entities. Thus, as long as the entity labels are known, raw text embeddings are clearly preferable.

5.5.1.2 Entity neighbourhoods

To understand the neighbourhood relation and proximity of embeddings, we look at most similar words by cosine similarity on the example entity *Barack Obama*. The proximity of embeddings shows how the models can be used to search for similar entities and terms. For models on raw text, we average the embeddings of words *barack* and *obama* and find the nearest neighbours to the average vector. A list of top five words for all models are shown in Table 5.10. Entity-based models are more focused on related entities, while embeddings on raw text tend to focus more on surrounding terms. rW2V, in particular, retrieves mostly misspelled versions of the entity name, or separate parts of the name. Since we are using the average of name parts to represent the entity in the vector space, it makes sense that the closest point to the average vector are name parts themselves. Top words for the raw model are the most similar words in terms of semantic similarity (mostly synonyms that can be substituted by the entity name in a text), yet, they do not give any further information about pos-



(a) rW2V (b) aW2V (c) VRS

Figure 5.2: t-SNE projections of the nearest neighbours of entity *Barack Obama*.

Table 5.10: Four nearest neighbours of the entity *Barack Obama* with their cosine similarity scores. Entity types include terms T, actors A, and locations L.

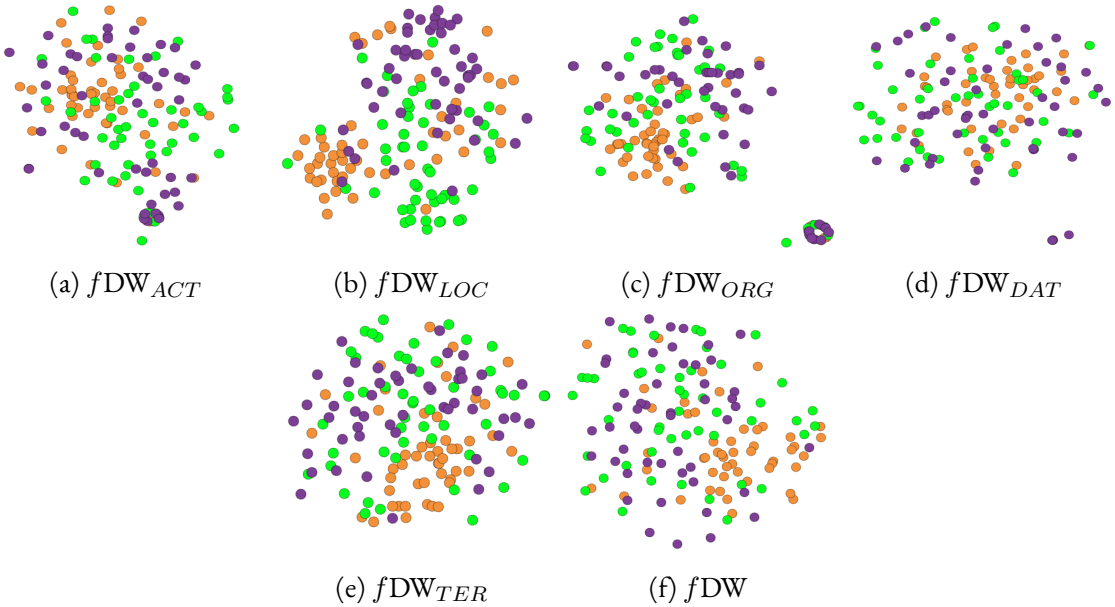
rW2V		rGLV		aW2V		aGLV	
T obama	0.90	T obama	0.99	A George W. Bush	0.76	T president	0.78
T barack	0.74	T barack	0.98	A Jimmy Carter	0.73	T administration	0.76
T obamaobama	0.68	T president	0.77	T barack obama	0.73	A George W. Bush	0.72
T obamathe	0.60	T administration	0.74	A Enrique Peña Nieto	0.67	T mr.	0.68
T bush	0.55	T elect	0.66	T then-vice	0.67	L White House	0.67

DW$_{id}$		DW$_{log}$		VRS	
L White House	0.88	L White House	0.88	L White House	0.87
T president	0.79	T president	0.82	T president	0.79
T presidency	0.76	A George W. Bush	0.78	L United States of America	0.76
T administration	0.75	T administration	0.78	A Donald Trump	0.75
A George W. Bush	0.74	T presidency	0.78	A Hillary Clinton	0.74

sible entity-entity relations. In similar fashion as the relatedness task, we observe that graph-based models on annotated text, VRS in particular, favours relatedness over similarity, as top retrieved words are related words that have some association to the entity in question.

The Figure 5.1 shows the neighbourhood of top ten closest words to *Barack Obama* for rW2V, aW2V, and VRS. We observe a similar trend in the 2-dimensional t-SNE projections, where word2vec on raw text tends to primarily identify words used synonymously on the raw corpus (i.e., variations of the entity name) and focuses on similarity. The word2vec on the annotated text, retrieves more entities, but also favours similarity, as the nearest neighbours are either other presidents or entities that share similar roles. In contrast, VERSE identifies related entities with different roles, such as administrative locations or the presidential candidates and the president elect in the 2016 U.S. election.

Overall, for tasks that require word similarity, word embeddings methods tend to produce the best result, while the graph-based models have better performance when it comes to identifying relations or associations between entities.



(a) $f\text{DW}_{ACT}$ (b) $f\text{DW}_{LOC}$ (c) $f\text{DW}_{ORG}$ (d) $f\text{DW}_{DAT}$

(e) $f\text{DW}_{TER}$ (f) $f\text{DW}$

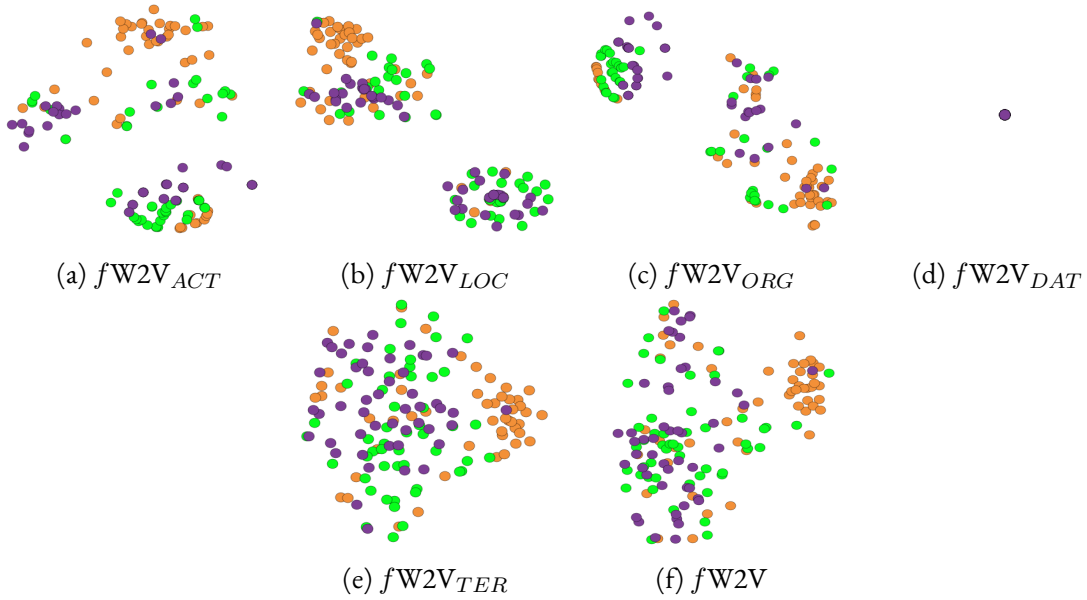
Figure 5.3: t-SNE projections of the embeddings for U.S. (purple), British (orange), and German (green) cities, for faceted deepwalk with logarithmic normalization using different components.

5.5.2 Analysis of faceted embeddings

To analyse separate components of faceted embeddings and investigate the semantics behind them, we consider visualizing the clustering task and analysing the neighbourhood for entities. In both cases, we look at full embeddings with all components concatenated and also analyse each component separately to understand how the different types of entities surrounding a word can contribute to its semantics and effect its relations to other entities.

5.5.2.1 Entity clustering

One way to understand which component is significant for the representation of entities is to asses which component clusters similar entities in the embedding space. To achieve this, we project the full and partial embedding of each component for the Cities dataset, which contains location entities, in a 2-dimensional space using t-SNE. Projections for $f\text{W2V}$ and $f\text{DW}_{log}$ are shown in Figure 5.3 and Figure 5.4, respectively. The subscript shows the embedding component that is used, e.g., $f\text{W2V}_{ACT}$ corresponds to faceted word2vec, using only the actor component and $f\text{W2V}$ indicates the full embedding. Since the performance of $f\text{DW}_{id}$ and $f\text{DW}_{log}$ are similar, we only illustrate the result for $f\text{DW}_{log}$. For both faceted models, it is not obvious what each component represents. None of the components of $f\text{DW}_{log}$ form any meaningful cluster, however, the distinction between three countries is somewhat visible for the location component. The projection of the organisation component, contains an outlier patch of only cities from U.S., but rest of city names are scattered with no pattern. The faceted word2vec clearly indicates that dates surrounding a city name are not a good feature for their distinction, as all names are mapped to the same point, which implies that no cities in this dataset had connections to terms in the co-occurrence graph. Two separate clusters are



(a) fW2V$_{ACT}$ (b) fW2V$_{LOC}$ (c) fW2V$_{ORG}$ (d) fW2V$_{DAT}$

(e) fW2V$_{TER}$ (f) fW2V

Figure 5.4: t-SNE projections of the embeddings for U.S. (purple), British (orange), and German (green) cities, for faceted word2vec using different components.

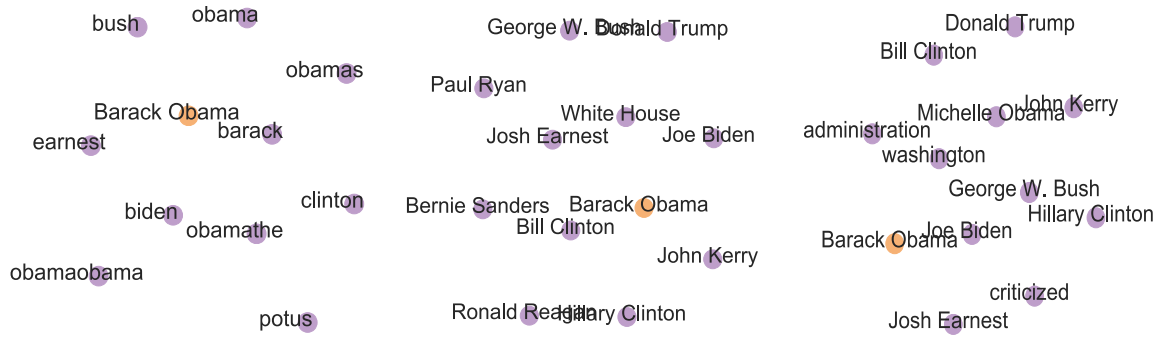
Table 5.11: Four nearest neighbours of the entity *Barack Obama* with cosine similarity scores for faceted models and embeddings on raw text. Entity types include terms T and actors A.

rW2V		rGLV		fW2V		fDW$_{id}$		fDW$_{log}$	
T obama	0.90	T obama	0.99	A Donald Trump	0.92	A Donald Trump	0.84	A Hillary Clinton	0.85
T barack	0.74	T barack	0.98	A Hillary Clinton	0.91	A Hillary Clinton	0.84	A Donald Trump	0.85
T obamaobama	0.68	T president	0.77	A Bill Clinton	0.89	T nation	0.83	A Bill Clinton	0.83
T obamathe	0.60	T administration	0.74	A Bernie Sanders	0.87	T force	0.83	A Josh Earnest	0.82
T bush	0.55	T elect	0.66	A George W. Bush	0.87	T freedom	0.82	T administration	0.82

visible for the location and organisation components of fW2V, but they are not separated by the county of origin, as both clusters are a mixture of all three countries. In general, for faceted models the visualization of the full embedding is closely related to the term component, which is the dominant part in these models.

5.5.2.2 Entity neighbourhoods

To analyse the proximity of embeddings based on different components, we investigate the most similar words by cosine similarity to the example actor *Barach Obama*. A list of top five words for faceted models with full embeddings and classical word embedding, rW2V and rGLV, is shown in Table 5.11. Similar to entity embeddings, the faceted models are more focused on entity relations, whereas the word embeddings on raw text retrieve synonymous terms. Although the fDW$_{id}$ tends to contain some terms in the top result, with the log normalization the result become close to fW2V. Both fW2V and fDW$_{log}$, mostly contain former presidents, the presidential candidate for the 2016 election and other democrat politicians.
The projection of the neighbourhood for rW2V, fDW$_{log}$, and fW2V is shown in Figure 5.5. While



(a) rW2V (b) fW2V (c) $f\text{DW}_{log}$

Figure 5.5: t-SNE projections of the nearest neighbours of entity *Barack Obama* for faceted embeddings.

Table 5.12: Four nearest neighbours of entity *Barack Obama* with cosine similarity scores. For separate components of fW2V and $f\text{DW}_{log}$. Entity types include terms T, actors A , and locations L, organisations O, dates D.

$f\text{W2V}_{TER}$		$f\text{W2V}_{ACT}$		$f\text{W2V}_{LOC}$		$f\text{W2V}_{ORG}$		$f\text{W2V}_{DAT}$	
L White House	0.97	L White House	0.98	T 0324	0.77	A White House	0.97	D 2016-01-26	0.64
T democratic	0.96	A Donald Trump	0.92	L St. John's Episcopal Church	0.76	A Donald Trump	0.93	D 2008-06-08	0.57
T presidential	0.95	L Washington, D.C.	0.92	D 2011-07-09	0.73	O U.S. House of Representatives	0.93	D 2005	0.56
A George W. Bush	0.94	A Ben Rhodes	0.91	L South Sudan	0.72	O U.S. Congress	0.93	D 1865-05	0.56
T republican	0.94	A Hillary Clinton	0.91	L Palazzo Grassi	0.71	A Hillary Clinton	0.93	D 1970	0.55

$f\text{DW}_{TER}$		$f\text{DW}_{ACT}$		$f\text{DW}_{LOC}$		$f\text{DW}_{ORG}$		$f\text{DW}_{DAT}$	
L Philadelphia	0.94	A Daniel M. Ashe	0.92	T decades	0.97	A Bill Clinton	0.97	T not	0.99
L Pennsylvania	0.94	A Hefty Stuart	0.90	T reported	0.97	L New York City	0.96	L Alabama	0.99
D 2016-07	0.92	A Elena Bromund	0.90	T statement	0.96	T according	0.95	T party	0.99
L New York City	0.92	A Brendan O'Connor	0.90	T rally	0.96	A Donald Trump	0.95	T timeline	0.98
O United States Congress	0.92	A Glenn Hutchins	0.90	T according	0.96	L Benghazi	0.95	T kept	0.98

faceted models tend to focus on entity-entity relations, rW2V retrieves miss-spelled versions of first- and last-names of former presidents of United States. Without the knowledge of the query word it is not obvious if the word *bush* refers to the entity *George W.Bush* or the whole *Bush* family or even shrubbery. On the other hand, faceted models, despite a few terms like *administration*, contain mostly associated politicians. Moreover, retrieved terms also tend to be more descriptive of the entity rather than synonymous.

In Table 5.12 we look at the top five results based on separate components for fW2V and $f\text{DW}_{log}$, where the subscript indicates the component that is used to find the nearest neighbour. $f\text{DW}_{id}$ produces similar results to $f\text{DW}_{log}$, thus, is not demonstrated. We observe an interesting trend in fW2V, where the type of the most similar entity to a word is dependent on the component type. Most similar entities to *Barack Obama* in the date components are dates themselves, including 2005 when his career in the United States Senate began. The same applies for locations, organisations and actors, where the majority of retrieved entities share the type of the component. For example, locations such as South Sudan in the location component or *United States Congress* for the organisation component. It appears that the topic of the attack on the embassy in *Benghazi* is dominant in the data as the raid happened on the 9th of July 2011. Important political actors related to this

event were former National Security Advisor *Ben Rhodes* and the former Secretary of State *Hillary Clinton*, who were all in the administration of *Barack Obama*. In case of $f\text{DW}_{log}$, the effect of component type is less visible, with only the actor component containing only actors in the top five. These actors have no obvious connection to *Barack Obama*, except *Glenn Hutchins*, who is a board member of Obama's foundation. The unrelated results of DeepWalk based models may lie in the structure of the co-occurrence graph and the type-restricted random walk, which was introduced to limit the context. Since we restricted the random walk to only edges that have a specific entity type as the end node for each component, we focus on very limited information. For example, to perform a type restricted random walk for actors from the start node of *Barack Obama*, we retrieve actors that are connected through other actors, which is not necessary helpful. Therefore, results, such as *Hefty Stuart*, the Australian cyclist, that are not a result of direct connection show up in the result.

5.6 Summary of Evalution Tasks

In this chapter, we reported the result of intrinsic evaluation on entity embeddings and faceted embeddings, we compare them against state-of-the-art word embedding methods on popular term-based intrinsic tasks. Furthermore, we investigated characteristics of each method through exploratory analysis and visualization.

In general, while word embeddings on raw text captures the similarity among words well, they tend to map synonymous words closer in the embedding space, while models on the annotated corpus (faceted and entity embeddings) focus on association among entities. Models that use the co-occurrence graphs from annotated text as input, in particular, focus on relatedness and are more entity oriented,. They also tend to produce better results on the entity-centric analogy task. Additionally, they are able retrieve associated entities with a neighbourhood search, based on the exploratory evaluation of their neighbourhoods.

Faceted models are similar to entity embedding, in terms of capturing relatedness and good performance on entity-centric analogy task. However, overall they perform poorly on most word similarity and categorization datasets. These models retrieve related entities with a neighbourhood search and have the extra advantage of separable components. Through studying the components separately, we show that for the most term-based intrinsic tasks, the most important entity type is term. Although components other than term are not significant for term-based evaluations, they can be used to search the neighbourhood of a word, based on entity types. fW2V is notably good at retrieving entities of a particular type with nearest neighbour search in the relevant subspace.

6 Conclusions

In this thesis, we investigated different techniques to jointly train embeddings for terms and entities on an annotated corpus with named entities. We considered the naive application of popular models, namely, word2vec and GloVe, to annotated texts, as well as embedding nodes of a co-occurrence graphs extracted from the annotated text. We used popular node embedding methods DeepWalk and VERSE and modified them to obtain entity embeddings. Furthermore, to analyse the effect of each entity type on the embedding space, we explored variations of GloVe, word2vec, and DeepWalk that separate the embedding space into different embedding types. These faceted models consist of separable components, where each component indicates the relation of a word to entities of a specific type and can be used for type-specific analysis. Moreover, we compared all proposed models to traditional word embeddings on a comprehensive set of term-focused evaluation tasks and performed entity-centric exploration to identify strengths and weaknesses of each approach.

6.1 Discussion

We found that even though training state-of-the-art word embeddings directly on annotated text is possible, their performance degrades in term-centric tasks, particularly in tasks that depend on relatedness. By contrast, graph-based embeddings, while bad at capturing similarity, perform better on the relatedness tasks. Moreover, they provide more insights into entity-entity relations and thus, work better for entity-centric problems. Graph-based embeddings, in particular, show great promise for entity-centric analogy tasks, but fail to form meaningful clusters of words, whereas normal word embeddings are able to cluster similar words and even multi-word entities better and perform well on purely term-based analogy tasks. Exploration of entity neighbourhoods for each model shows that proximity of words with similar meanings is better for regular word embeddings, as the neighbourhood shows only synonymous or descriptive words. On the other hand, models on annotated text and graph-based embeddings, in particular, tend to map related and associated entities closer in embedding space and can be used for searching for related entities. In general, the usefulness of entity embeddings is very dependent on the problem at hand, and should not be used blindly in place of word embeddings.

Since faceted models use annotated text as input, they share many characteristics with the entity embeddings. Faceted models tend to favour relatedness over similarity and show promising results in entity-centric analogy tasks. However, dividing the embedding space by removing certain entity types during training has its drawbacks. The performance significantly degrades in comparison to

normal word embeddings on raw text, which we consider due to a lack of information to learn from. To train each component, the vocabulary is limited to a certain entity type, which limits the context of the word and makes it more difficult for the model to learn a useful representation. Approaches based on neural networks, however, require larger amounts of data and their performance is strongly coupled to data availability. Hence, limiting the context of a word to a certain type of entity reduces the training data set, resulting in a lower performance. We hypothesize that with a more involved definition of context for these models, where the limitation of context does not limit the data for the model to learn from, we might observe a boost in performance. Despite the decrease in performance, faceted models show promise in entity-centric search, where each individual components can be used to limit the search space to a specific type. Moreover, similar to entity embeddings, they tend to map related entities closer together rather than synonymous words. Furthermore, by independent study of each component of faceted models, we found that surrounding terms are most significant when it comes to term-based relatedness and similarity tasks, while for clustering similar entities the component corresponding to the entity type tends to produce better results.

6.2 Future Work

Although the comparison of entity-based models with word embeddings on term-based datasets sheds some light onto their differences and potential use-cases, true performance of these models can only be evaluated if entity-centric evaluation datasets are created. As future work, creating a dataset that contains named entities for common intrinsic tasks for word embedding and designing specific intrinsic tasks for entity-centric evaluation will support the evaluation of future entity-based models to efficiently evaluate their methods.

Because of the ability of entity-based models to capture relatedness, a temporal analysis of the evolution of entity-entity relations in a dataset with a temporal aspect, such as news streams, is possible. Similar approaches have been used to study the transition in meaning for a word over time, which can be utilized to investigate the evolution of entity-entity relations. Moreover, separate components of faceted embeddings allow for a more detailed study, where the relation of an entity to associated locations or actors are analysed separately.

Overall, considering all weaknesses and strength of faceted and entity-based models, we see potential applications for entity embeddings and faceted models in entity-centric tasks that benefit from relatedness relations, such as improved query expansion based on related entities. Document ranking can also benefit from a vector representation of entity and words, especially when the document contains related entities to query words. Faceted models can be used to create more flexible search criteria based on entity types, which is beneficial for entity ranking and search engines. Tasks that use named entities as part of their pipeline can also benefit from entity embeddings, such as named entity recognition and disambiguation, which we consider to be the most promising future research directions and downstream tasks.

Acronyms

aGLV	The GloVe model on annotated data
aW2V	Word2vec model on annotated data
fDW	Facetted DeepWalk on word co-occurrence graph
fGLV	Facetted GloVe
$f\text{GLV}_{sep}$	Facetted GloVe with separate cost functions
$f\text{GLV}_{uni}$	Facetted GloVe with unified cost functions
fW2V	Facetted word2vec
rGLV	The GloVe model on raw data
rW2V	Word2vec model on raw data
BFS	Breadth-first Sampling
CBOW	Continuous bag of words
DFS	Depth-first Sampling
DW	DeepWalk on word co-occurrence graph
DW_{id}	DeepWalk with identity mapping
DW_{log}	DeepWalk with logarithmic mapping
GD	Gradient Descent
GloVe	Global Vector embeddings
IR	Information Retrial
KL	Kullback-Leibler divergence
LINE	Large-scale Information Network Embedding
LOAD	Locations,Organisations,Actors,Dates
NLP	Natural language processing
POS	Part of speech
PPR	Personalized PageRank
SVD	Singular value decomposition
VERSE	Versatile Graph Embeddings from Similarity Measures
VRS	VERSE on word co-occurrence graph

Glossary

LOC	location
ACT	Actor
$a_i^{[j]}$	Activation value of the neuron i in layer j
A	Set of all actors
X_{ij}	Co-occurrence count of words i and j in the corpus
C	Set of all context words
DAT	Date
D	Set of all dates
δ	LOAD distance function
e_{ij}	Edge weight between nodes i and j
E	Set of all weighted edges of the word co-occurrence graph
f	Center (focal) word
G	The word co-occurrence graph
J	Cost function
L	Set of all locations
N	Set of all named entities
$\overline{N}$	Set of all words, which are not named entities
ORG	Organisation
O	Set of all organisations
Q	The set of all focal and context pairs extracted from the text
Sim_E	Similarity measure between embeddings of the graph nodes
Sim_G	Similarity measure between nodes of the graph
TER	Term
T	Set of all terms
$\theta_{nl}^{(j)}$	The weight controlling the function mapping from layer j to layer $j+1$ between unit n in layer j and l in $j+1$
V_e	The focal vocabulary
V	The complete vocabulary, set of all the words
w_i	A word embedding for word i
$y^{(i)}$	True label
$\hat{y}^{(i)}$	Predicted output

Bibliography

Agirre, E., E. Alfonseca, K. B. Hall, J. Kravalova, M. Pasca, and A. Soroa (2009). "A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches". In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA*, pp. 19–27. URL: http://www.aclweb.org/anthology/N09-1003.

Al-Rfou, R., V. Kulkarni, B. Perozzi, and S. Skiena (2015). "POLYGLOT-NER: Massive Multilingual Named Entity Recognition". In: *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, pp. 586–594. DOI: 10.1137/1.9781611974010.66. URL: https://doi.org/10.1137/1.9781611974010.66.

Bakarov, A. (2018). "A Survey of Word Embeddings Evaluation Methods". *CoRR* abs/1801.09536. arXiv: 1801.09536. URL: http://arxiv.org/abs/1801.09536.

Bansal, M., K. Gimpel, and K. Livescu (2014). "Tailoring Continuous Word Representations for Dependency Parsing". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pp. 809–815. URL: http://aclweb.org/anthology/P/P14/P14-2131.pdf.

Baroni, M., S. Evert, and A. Lenci, eds. (2008). *Proceedings of the ESSLLI Workshop on Distributional Lexical Semantics Bridging the Gap Between Semantic Theory and Computational Simulations*.

Bengio, Y., R. Ducharme, and P. Vincent (2000). "A Neural Probabilistic Language Model". In: *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pp. 932–938. URL: http://papers.nips.cc/paper/1839-a-neural-probabilistic-language-model.

Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). "Enriching Word Vectors with Subword Information". *TACL* 5, pp. 135–146. URL: https://transacl.org/ojs/index.php/tacl/article/view/999.

Bruni, E., N. Tran, and M. Baroni (2014). "Multimodal Distributional Semantics". *J. Artif. Intell. Res.* 49, pp. 1–47. DOI: 10.1613/jair.4135. URL: https://doi.org/10.1613/jair.4135.

Cai, H., V. W. Zheng, and K. C. Chang (2018). "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications". *IEEE Trans. Knowl. Data Eng.* 30:9, pp. 1616–1637. DOI: 10.1109/TKDE.2018.2807452. URL: https://doi.org/10.1109/TKDE.2018.2807452.

Chen, D. and C. D. Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Pro-*

cessing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pp. 740–750. URL: http://aclweb.org/anthology/D/D14/D14-1082.pdf.

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa (2011). "Natural Language Processing (Almost) from Scratch". *Journal of Machine Learning Research* 12, pp. 2493–2537. URL: http://dl.acm.org/citation.cfm?id=2078186.

Cover, T. M. and J. A. Thomas (2006). *Elements of information theory (2. ed.)* Wiley. ISBN: 978-0-471-24195-9.

Diaz, F., B. Mitra, and N. Craswell (2016). "Query Expansion with Locally-Trained Word Embeddings". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. URL: http://aclweb.org/anthology/P/P16/P16-1035.pdf.

Dong, Y., N. V. Chawla, and A. Swami (2017). "metapath2vec: Scalable Representation Learning for Heterogeneous Networks". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 135–144. DOI: 10.1145/3097983.3098036. URL: http://doi.acm.org/10.1145/3097983.3098036.

Durme, B. V., P. Rastogi, A. Poliak, and M. P. Martin (2017). "Efficient, Compositional, Order-sensitive n-gram Embeddings". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pp. 503–508. URL: https://aclanthology.info/papers/E17-2081/e17-2081.

Eshel, Y., N. Cohen, K. Radinsky, S. Markovitch, I. Yamada, and O. Levy (2017). "Named Entity Disambiguation for Noisy Text". In: *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Vancouver, Canada, August 3-4, 2017*, pp. 58–68. DOI: 10.18653/v1/K17-1008. URL: https://doi.org/10.18653/v1/K17-1008.

Fang, W., J. Zhang, D. Wang, Z. Chen, and M. Li (2016). "Entity Disambiguation by Knowledge and Text Jointly Embedding". In: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pp. 260–269. URL: http://aclweb.org/anthology/K/K16/K16-1026.pdf.

Faruqui, M., Y. Tsvetkov, D. Yogatama, C. Dyer, and N. A. Smith (2015). "Sparse Overcomplete Word Vector Representations", pp. 1491–1500. URL: http://aclweb.org/anthology/P/P15/P15-1144.pdf.

Freilich, S., A. Kreimer, I. Meilijson, U. Gophna, R. Sharan, and E. Ruppin (2010). "The Large-scale Organization of the Bacterial Network of Ecological Co-occurrence Interactions". *Nucleic Acids Research* 38:12, pp. 3857–3868. DOI: 10.1093/nar/gkq118. eprint: /oup/backfile/content_public/journal/nar/38/12/10.1093_nar_gkq118/1/gkq118.pdf. URL: http://dx.doi.org/10.1093/nar/gkq118.

Goldberg, Y. (2016). "A Primer on Neural Network Models for Natural Language Processing". *J. Artif. Intell. Res.* 57, pp. 345–420. DOI: 10.1613/jair.4992. URL: https://doi.org/10.1613/jair.4992.

Goldberg, Y. and O. Levy (2014). "word2vec Explained: Deriving Mikolov et al.'s Negative-sampling Word-embedding Method". *CoRR* abs/1402.3722. arXiv: 1402.3722. URL: http://arxiv.org/abs/1402.3722.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goyal, P. and E. Ferrara (2018). "Graph Embedding Techniques, Applications, and Performance: A Survey". *Knowl.-Based Syst.* 151, pp. 78–94. DOI: 10.1016/j.knosys.2018.03.022. URL: https://doi.org/10.1016/j.knosys.2018.03.022.

Grover, A. and J. Leskovec (2016). "node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 855–864. DOI: 10.1145/2939672.2939754. URL: http://doi.acm.org/10.1145/2939672.2939754.

Guo, Z. and D. Barbosa (2018). "Robust Named Wntity Disambiguation with Random Walks". *Semantic Web* 9:4, pp. 459–479. DOI: 10.3233/SW-170273. URL: https://doi.org/10.3233/SW-170273.

Gutmann, M. and A. Hyvärinen (2012). "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics". *Journal of Machine Learning Research* 13, pp. 307–361. URL: http://dl.acm.org/citation.cfm?id=2188396.

Hashimoto, K., P. Stenetorp, M. Miwa, and Y. Tsuruoka (2015). "Task-Oriented Learning of Word Embeddings for Semantic Relation Classification". In: *Proceedings of the 19th Conference on Computational Natural Language Learning, CoNLL 2015, Beijing, China, July 30-31, 2015*, pp. 268–278. URL: http://aclweb.org/anthology/K/K15/K15-1027.pdf.

Haykin, S. S. (2009). *Neural networks and learning machines*. Third. Pearson Education, Upper Saddle River, NJ.

Hill, F., R. Reichart, and A. Korhonen (2015). "SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation". *Computational Linguistics* 41:4, pp. 665–695. DOI: 10.1162/COLI_a_00237. URL: https://doi.org/10.1162/COLI_a_00237.

Hill, F., K. Cho, A. Korhonen, and Y. Bengio (2016). "Learning to Understand Phrases by Embedding the Dictionary". *TACL* 4, pp. 17–30. URL: https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/711.

Jurafsky, D. and J. H. Martin (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. Prentice Hall PTR, Upper Saddle River, NJ, USA. ISBN: 0130950696.

– (2016). *Speech and Language Processing*.

Kiefer, J., J. Wolfowitz et al. (1952). "Stochastic Estimation of the Maximum of a Regression Function". *The Annals of Mathematical Statistics* 23:3, pp. 462–466.

Kolb, P. (2009). "Experiments on the difference between semantic similarity and relatedness". In: *Proceedings of the 17th Nordic Conference of Computational Linguistics, NODALIDA 2009,*

Odense, Denmark, May 14-16, 2009, pp. 81–88. URL: https://aclanthology.info/papers/W09-4613/w09-4613.

Kuzi, S., A. Shtok, and O. Kurland (2016). "Query Expansion Using Word Embeddings". In: *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pp. 1929–1932. DOI: 10.1145/2983323.2983876. URL: http://doi.acm.org/10.1145/2983323.2983876.

Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). "Neural Architectures for Named Entity Recognition". In: *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pp. 260–270. URL: http://aclweb.org/anthology/N/N16/N16-1030.pdf.

Larson, R. R. (2010). *Introduction to information retrieval*. Wiley Online Library.

Levy, O. and Y. Goldberg (2014a). *Embeddings with Arbitrary Contexts Code*. URL: https://bitbucket.org/yoavgo/word2vecf.

Levy, O. and Y. Goldberg (2014b). "Dependency-Based Word Embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pp. 302–308. URL: http://aclweb.org/anthology/P/P14/P14-2050.pdf.

– (2014c). "Neural Word Embedding as Implicit Matrix Factorization". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2177–2185. URL: http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.

Ling, W., C. Dyer, A. W. Black, and I. Trancoso (2015). "Two/Too Simple Adaptations of Word2Vec for Syntax Problems". In: *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pp. 1299–1304. URL: http://aclweb.org/anthology/N/N15/N15-1142.pdf.

Liu, Q., H. Huang, Y. Gao, X. Wei, Y. Tian, and L. Liu (2018). "Task-oriented Word Embedding for Text Classification". In: *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pp. 2023–2032. URL: https://aclanthology.info/papers/C18-1172/c18-1172.

Luong, T., R. Socher, and C. D. Manning (2013). "Better Word Representations with Recursive Neural Networks for Morphology". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*, pp. 104–113. URL: http://aclweb.org/anthology/W/W13/W13-3512.pdf.

Ma, Y., J. Kim, B. Bigot, and M. T. Khan (2016). "Feature-enriched Word Embeddings for Named Entity Recognition in Open-domain Conversations". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*,

pp. 6055–6059. DOI: 10.1109/ICASSP.2016.7472840. URL: https://doi.org/10.1109/ICASSP.2016.7472840.

Maaten, L. v. d. and G. Hinton (2008). "Visualizing Data Using t-SNE". *Journal of Machine Learning Research* 9:Nov, pp. 2579–2605.

Mandelbaum, A. and A. Shalev (2016). "Word Embeddings and Their Use In Sentence Classification Tasks". *CoRR* abs/1610.08229. arXiv: 1610.08229. URL: http://arxiv.org/abs/1610.08229.

Manning, C. D. and H. Schütze (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA. ISBN: 0-262-13360-1.

Manning, C. D., P. Raghavan, and H. Schütze (2008). *Introduction to Information Retrieval*. Cambridge University Press. ISBN: 978-0-521-86571-5.

Mihalcea, R. F. and D. R. Radev (2011). *Graph-based Natural Language Processing and Information Retrieval*. 1st. Cambridge University Press, New York, NY, USA. ISBN: 0521896134, 9780521896139.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013a). "Efficient Estimation of Word Representations in Vector Space". *CoRR* abs/1301.3781. arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781.

Mikolov, T., W. Yih, and G. Zweig (2013b). "Linguistic Regularities in Continuous Space Word Representations". In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pp. 746–751. URL: http://aclweb.org/anthology/N/N13/N13-1090.pdf.

Mirza, P. and S. Tonelli (2016). "On the Contribution of Word Embeddings to Temporal Relation Classification". In: *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pp. 2818–2828. URL: http://aclweb.org/anthology/C/C16/C16-1265.pdf.

Mitchell, J. and M. Lapata (2010). "Composition in Distributional Models of Semantics". *Cognitive Science* 34:8, pp. 1388–1429. DOI: 10.1111/j.1551-6709.2010.01106.x. URL: https://doi.org/10.1111/j.1551-6709.2010.01106.x.

Nadeau, D. and S. Sekine (2007). "A Survey of Named Entity Recognition and Classification". *Lingvisticae Investigationes* 30:1, pp. 3–26.

Nastase, V., R. Mihalcea, and D. R. Radev (2015). "A Survey of Graphs in Natural Language Processing". *Natural Language Engineering* 21:5, pp. 665–698. DOI: 10.1017/S1351324915000340. URL: https://doi.org/10.1017/S1351324915000340.

Park, S., J. Bak, and A. Oh (2017). "Rotated Word Vector Representations and their Interpretability". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pp. 401–411. URL: https://aclanthology.info/papers/D17-1041/d17-1041.

Pennington, J., R. Socher, and C. D. Manning (2014). "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Process-*

ing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pp. 1532–1543. URL: http://aclweb.org/anthology/D/D14/D14-1162.pdf.

Perozzi, B., R. Al-Rfou, and S. Skiena (2014a). "DeepWalk: Online Learning of Social Representations". In: *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pp. 701–710. DOI: 10.1145/2623330.2623732. URL: http://doi.acm.org/10.1145/2623330.2623732.

Perozzi, B., R. Al-Rfou, and S. Skiena (2014b). *DeepWalk Package in Python*. URL: https://github.com/phanein/deepwalk.

Radinsky, K., E. Agichtein, E. Gabrilovich, and S. Markovitch (2011). "A Word at a Time: Computing Word Relatedness Using Temporal Semantic Analysis". In: *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pp. 337–346. DOI: 10.1145/1963405.1963455. URL: http://doi.acm.org/10.1145/1963405.1963455.

Rohde, D. L., L. M. Gonnerman, and D. C. Plaut (2006). "An improved model of semantic similarity based on lexical co-occurrence". *Communications of the ACM* 8:627-633, p. 116.

Rousseau, F. and M. Vazirgiannis (2013). "Graph-of-word and TW-IDF: New Approach to Ad hoc IR". In: *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pp. 59–68. DOI: 10.1145/2505515.2505671. URL: http://doi.acm.org/10.1145/2505515.2505671.

Rubenstein, H. and J. B. Goodenough (1965). "Contextual Correlates of Synonymy". *Commun. ACM* 8:10, pp. 627–633. DOI: 10.1145/365628.365657. URL: http://doi.acm.org/10.1145/365628.365657.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning Representations by Back-propagating Errors". *Nature* 323:6088, p. 533.

Sammut, C. and G. I. Webb, eds. (2017). *Encyclopedia of Machine Learning and Data Mining*. Springer. ISBN: 978-1-4899-7685-7. DOI: 10.1007/978-1-4899-7687-1. URL: https://doi.org/10.1007/978-1-4899-7687-1.

Schnabel, T., I. Labutov, D. M. Mimno, and T. Joachims (2015). "Evaluation Methods for Unsupervised Word Embeddings". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 298–307. URL: http://aclweb.org/anthology/D/D15/D15-1036.pdf.

Shen, W., J. Wang, and J. Han (2015). "Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions". *IEEE Trans. Knowl. Data Eng.* 27:2, pp. 443–460. DOI: 10.1109/TKDE.2014.2327028. URL: https://doi.org/10.1109/TKDE.2014.2327028.

Shlens, J. (2014). "A Tutorial on Principal Component Analysis". *CoRR* abs/1404.1100. arXiv: 1404.1100. URL: http://arxiv.org/abs/1404.1100.

Siencnik, S. K. (2015). "Adapting word2vec to Named Entity Recognition". In: *Proceedings of the 20th Nordic Conference of Computational Linguistics, NODALIDA 2015, May 11-13, 2015, In-*

stitute of the Lithuanian Language, Vilnius, Lithuania, pp. 239–243. URL: http://aclweb.org/anthology/W/W15/W15-1830.pdf.

Spitz, A. and M. Gertz (2016). "Terms over LOAD: Leveraging Named Entities for Cross-Document Extraction and Summarization of Events". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pp. 503–512. DOI: 10.1145/2911451.2911529. URL: http://doi.acm.org/10.1145/2911451.2911529.

– (2018). "Entity-Centric Topic Extraction and Exploration: A Network-Based Approach". In: *Advances in Information Retrieval - 40th European Conference on IR Research, ECIR 2018, Grenoble, France, March 26-29, 2018, Proceedings*, pp. 3–15. DOI: 10.1007/978-3-319-76941-7_1. URL: https://doi.org/10.1007/978-3-319-76941-7_1.

Strötgen, J. and M. Gertz (2013). "Multilingual and Cross-domain Temporal Tagging". *Language Resources and Evaluation* 47:2, pp. 269–298. DOI: 10.1007/s10579-012-9179-y. URL: https://doi.org/10.1007/s10579-012-9179-y.

Subramanian, A., D. Pruthi, H. Jhamtani, T. Berg-Kirkpatrick, and E. H. Hovy (2018). "SPINE: SParse Interpretable Neural Embeddings". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17433.

Tang, J., M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei (2015). "LINE: Large-scale Information Network Embedding". In: *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pp. 1067–1077. DOI: 10.1145/2736277.2741093. URL: http://doi.acm.org/10.1145/2736277.2741093.

Toral, A., R. Muñoz, and M. Monachini (2008). "Named Entity WordNet". In: *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*. URL: http://www.lrec-conf.org/proceedings/lrec2008/summaries/188.html.

Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network". In: *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*. URL: http://aclweb.org/anthology/N/N03/N03-1033.pdf.

Tsitsulin, A., D. Mottin, P. Karras, and E. Müller (2018a). *VERSE Package in C++:* URL: https://github.com/xgfs/verse.

Tsitsulin, A., D. Mottin, P. Karras, and E. Müller (2018b). "VERSE: Versatile Graph Embeddings from Similarity Measures". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pp. 539–548. DOI: 10.1145/3178876.3186120. URL: http://doi.acm.org/10.1145/3178876.3186120.

Yamada, I., H. Shindo, H. Takeda, and Y. Takefuji (2016). "Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation". In: *Proceedings of the 20th SIGNLL*

Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016, pp. 250–259. URL: http://aclweb.org/anthology/K/K16/K16-1025.pdf.

Yang, B., W. Yih, X. He, J. Gao, and L. Deng (2014). "Embedding Entities and Relations for Learning and Inference in Knowledge Bases". *CoRR* abs/1412.6575. arXiv: 1412.6575. URL: http://arxiv.org/abs/1412.6575.

Yin, R., H. Bredin, and C. Barras (2018). "Neural Speech Turn Segmentation and Affinity Propagation for Speaker Diarization". In: *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*. Pp. 1393–1397. DOI: 10.21437/Interspeech.2018-1750. URL: https://doi.org/10.21437/Interspeech.2018-1750.

Yin, W. and H. Schütze (2014). "An Exploration of Embeddings for Generalized Phrases". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Student Research Workshop*, pp. 41–47. URL: http://aclweb.org/anthology/P/P14/P14-3006.pdf.

Yoon, T., S. Myaeng, H. Woo, S. Lee, and S. Kim (2018). "On Temporally Sensitive Word Embeddings for News Information Retrieval". In: *Proceedings of the Second International Workshop on Recent Trends in News Information Retrieval co-located with 40th European Conference on Information Retrieval (ECIR 2018), Grenoble, France, March 26, 2018*. Pp. 51–56. URL: http://ceur-ws.org/Vol-2079/paper11.pdf.

Zamani, H. and W. B. Croft (2017). "Relevance-based Word Embedding". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pp. 505–514. DOI: 10.1145/3077136.3080831. URL: http://doi.acm.org/10.1145/3077136.3080831.

Zhang, D., J. Yin, X. Zhu, and C. Zhang (2018). "Network Representation Learning: A Survey". *IEEE Transactions on Big Data*.

Zhong, G., L.-N. Wang, X. Ling, and J. Dong (2016). "An Overview on Data Representation Learning: From Traditional Feature Learning to Recent Deep Learning". *The Journal of Finance and Data Science* 2:4, pp. 265–278.

Zhou, D., S. Niu, and S. Chen (2018). "Efficient Graph Computation for Node2Vec". *CoRR* abs/1805.00280. arXiv: 1805.00280. URL: http://arxiv.org/abs/1805.00280.