From Word Vectors to Sentence Vectors

Reyhaneh Hashempour

MSc. Dissertation

Department of Artificial Intelligence

Faculty of Information and Communication Technology

University of Malta

2018

Supervisor(s):

Lonneke van der Plas, Institute of Linguistics and Language Technology,

University of Malta

Arantxa Otegi, Faculty of Informatics, University of the Basque

Country

Submitted in partial fulfilment of the requirements for the Degree of

European Master of Science in Human Language Science and Technology

M.Sc. (HLST)

FACULTY OF INFORMATION AND

COMMUNICATION TECHNOLOGY

UNIVERSITY OF MALTA

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I, the undersigned, declare that the Master’s dissertation submitted is my own work, except where acknowledged and referenced.

I understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Student Name:	*Reyhaneh Hashempour*
Course Code	CSA5310 HLST Dissertation
Title of work:	*From Word Vectors to Sentence Vectors*

Signature of Student:

Date: September 30th , 2018

Supervisors

Lonneke van der Plas

Institute of Linguistics and Language Technology

University of Malta

and

Arantxa Otegi

Faculty of Informatics

University of the Basque Country

Abstract

Helping computers understand the meaning of sentences is one of the most challenging tasks in Natural Language Processing (NLP). According to the principle of compositionality, the meaning of a whole (e.g. a sentence) is a function of its parts and the way they are combined. Sentences are composed of words; thus the first step to create proper representations for sentences is to build appropriate representations for words.

We argue that the type of context and the size of corpus affect the quality of the resulting word embeddings. To verify this, we build three different semantic spaces using three different contexts (bag-of-words, syntactic, and character n-gram) and two corpora of different sizes (British National Corpus and Wikipedia). We evaluate the resulting embeddings in various tasks qualitatively and quantitatively. Then we define three composition models (summation/averaging, concatenation and multiplication) to build sentence representations using the previous word embeddings. These models are evaluated in two different tasks (phrase and sentence similarity) to see which model of word embeddings along with which composition model perform best.

The results suggest that among the three types of context, the syntactic one needs the largest corpus, while the character n-gram context the smallest corpus to produce quality word embeddings, while the bag-of-words is the fastest to build. Moreover, representing a sentence by averaging the vectors of its constituent words yields the best results in phrase and sentence level similarity tasks compared to the other methods of composition (i.e. concatenation and multiplication). Furthermore, the character n-gram creates the best embeddings to represent the sentence’s constituent words in the task of paraphrase detection, especially when the size of the corpus is small.

Contents

1 Introduction

Distributional Semantic Models (DSMs) are computational models to represent words. In these models, each word is represented by a vector in a space where semantically related words are close and unrelated words are distant. Despite the variations in building these models, the theories and assumptions behind them are the same. All these models are based on the Distributional Hypothesis (Harris, 1954) and all adopt a geometric approach toward natural language semantics.

Distributional Hypothesis (DH) is based on the idea that words occurring in similar contexts tend to have a similar meaning. This hypothesis is appealing to computational linguists since it provides a solid theoretical foundation to automatically extract the semantic relations between words by considering their real use without any human intervention and without any a priori knowledge about semantics. Moreover, it offers the possibility to create a semantic space where a geometric metaphor can be used for meaning representation.

The core idea to adopt a geometric approach to natural language semantics is that semantic similarity can be represented as proximity in n-dimensional space. According to Sahlgren (2006) employing spatial proximity to represent semantic similarity is not accidental. In fact, it is very natural and intuitive to us since we are embodied beings who use our spatiotemporal knowledge to make sense of the world and conceptualize the abstract concepts. Many aspects of semantics, especially lexical semantics require a notion of distance. For example, when comparing the meaning of words *dog*, *cat*, and *car*, we would say the meaning of the word *cat* is closer to the meaning of the word *dog* than the meaning of the word *car*.

Exploiting the aforementioned ideas, DSMs try to represent a word in the form of a finite-dimensional vector of real numbers in a vector space. The dimensions of this vector are the word's context items (e.g. co-occurring words) and the coordinates are a measure of the association between the word and the context items (e.g. the co-occurrence count). Different methods have been suggested by the scholars for building these models and although all of them are based on the same theory (i.e. DH), they are divided into two main categories: count-based models and predictive models[1]. The following section discusses each category in detail.

1.2 Distributional Semantic Models: Count-based

As mentioned earlier, creating a vector space of word meaning begins by extracting distributional information from a text corpus. Based on the DH, the distribution of a word in a corpus or its co-occurrences with other words is a good indication of its meaning. Extracting these co-occurrences from a corpus is an easy task for computers. Consider the following toy corpus of 4 sentences:

I heated the soup on the stove.

I heated the pizza in the microwave.

Wear warmer clothes, it is cold.

Were you wearing these shoes at the party?!

The co-occurrence count at the sentence level i.e., every word is considered to be the context of another word if they are in the same sentence, can be represented in the form of the matrix in figure 1.1. This matrix is built after filtering out the functional words (e.g. in) and lemmatizing the remaining words.

[1] This categorization is commonplace in the literature since Baroni et al.'s paper "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors" in 2013.

	(	heat	soup	pizza	wear	warm	cold	party	)
stove		1	1	0	0	0	0	0	
microwave		1	0	1	0	0	0	0	
clothes		0	0	0	1	1	1	0	
shoes		0	0	0	1	0	0	1	

Figure 1.1. A small example corpus and term vocabulary with the corresponding term-term matrix, with term-frequency as the vector coefficients. Each sentence provides a contextual window, and the sentences are assumed to have been lemmatized when creating the matrix.

The matrix shown in Figure 1.1 is the matrix of co-occurrence counts of words in the corpus. The rows of the matrix represent the target words, i.e. the words that we want to find the representations for (here: stove, microwave, clothes, and shoes), and the columns are the features that are important for deriving the word's representation which are the lemmatized content words. Every row in this matrix is the vector representation of the corresponding word. Thus, the vectors for stove, microwave, and shoes are as follows:

$\overrightarrow{Stove}$= [1 1 0 0 0 0 0]; $\overrightarrow{\text{mıcrowave}}$= [1 0 1 0 0 0 0]; $\overrightarrow{Shoes}$= [0 0 0 1 0 0 1]

In order to find the similarity between these words, we could calculate the similarity between their vectors. A commonly used method for vector similarity is cosine similarity, i.e. we just need to calculate the cosine of the angle between their vectors. This value ranges from 0 to 1 where 0 suggests non-similarity and 1 perfect similarity between two words. Here is the formula to calculate the cosine similarity:

$$\cos(\theta) = \frac{\vec{a}.\vec{b}}{\|\vec{a}\|.\|\vec{b}\|}$$

, where θ is the angle between vectors a and b, and $\|\vec{a}\|\ and \|\vec{b}\|$ are the size of vectors a and b respectively.

Cosine $(\overrightarrow{Stove}, \overrightarrow{\text{microwave}}) = 0.5$

Cosine $(\overrightarrow{Stove}, \overrightarrow{Shoes}) = 0.0$

The results show 50% similarity between stove and microwave and 0.0% similarity between stove and shoes for this tiny corpus of 4 sentences.

1.3 Distributional Semantic Models: Predictive

Predictive models create word vectors that are the results of training a neural network for a particular task. For example, Bengio et al. (2003) proposed a neural network model in which the task was to predict the next word given the previous words in a sentence. Similar to the count-based method, words are again represented by their contexts. The only difference is the way the problem is formulated. One of the commonly used algorithms in this approach is the one introduced by Mikolov et al. (2013) which acquires word embeddings by solving a co-occurrence prediction task. This algorithm has two flavors: Continuous Bag of Words (CBOW) and SkipGram. We discuss these two models, CBOW and SkipGram, in more detail in the following parts with the emphasis on SkipGram. SkipGram model is finally used to build word vectors for the purpose of this work.

As mentioned before, two different approaches can be taken to build word embeddings using co-occurrence prediction: CBOW and SkipGram. In CBOW, the target word is predicted given the context and in SkipGram, the context is predicted given the word. Needless to say, in both cases, the context is composed of the words before and after the target word.

Figure 1.2 illustrates the two models. The left side of the figure predicts the target word, given the words on its left and right sides. In other words, it predicts "*on*", given the words "*cat*", "*sat*", "*the*", and "*mat*" in its context. As mentioned before, this model is called CBOW. The right side of the figure demonstrates the other model which is SkipGram where the context of a word is predicted given the word. Here, the model learns to predict the context words "*cat*", "*sat*", "*the*", and "*mat*" given the word "*on*".

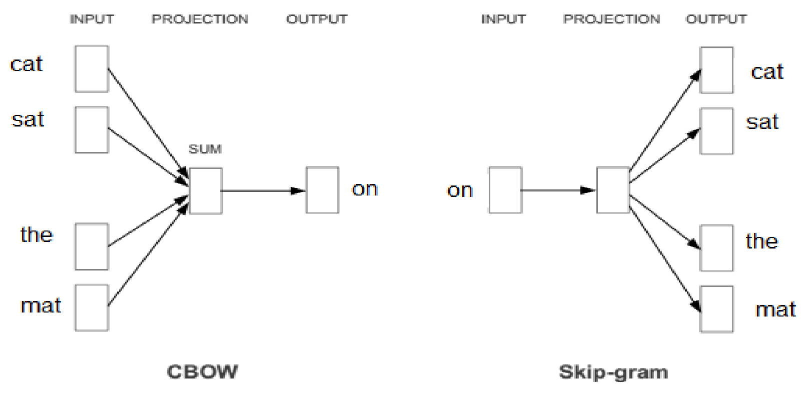


Figure 1.2. The left side of the figure shows CBOW which predicts the target word, given the words on its left and right sides. The right side of the figure demonstrates the other model which is SkipGram where the context of a word is predicted given the word.

Now, let's have a closer look at the SkipGram model.

1.3.1 Mathematical Description of the SkipGram Algorithm

In this part, the model is described following the explanation provided by Goldberg & Levy (2014), and we show how this model offers a possibility of incorporating contexts different from the linear bag of words (BOW).

Before starting the explanation, it is worth mentioning that in this model, the vectors of the words are different from those of the contexts. This assumption has to be made for the model to work. For example, let's assume that the vector of *dog* as a word is the same as the vector of *dog* as a context; since a word hardly appear in its own context, the model should assign a low probability to $p(dog|dog)$which entails a low value to $v \times v$ (dot products of two vectors) which is not possible since the words are totally similar.

Considering the aforementioned assumption, SkipGram works like this: each word $\omega \epsilon W$ is attributed with a vector $v_{\omega} \epsilon R^{d}$ and each context $c \epsilon C$ is represented as $v_{c} \epsilon R^{d}$ where W is the words vocabulary and C is the contexts vocabulary, and d is the embedding dimensionality (the size of word vectors). The elements in the vectors are latent and learned during the training period. In other words, their values are determined so that the dot product of v_ω and v_c $(v_\omega \times v_c)$ for a proper word-context pair is maximized.

Consider (ω, c) as a word-context pair; $p(D = 1|\omega, c)$ denotes the probability that the pair come from the data and $p(D = 0|\omega, c) = 1 - p(D = 1|\omega, c)$ denotes the very opposite. The distribution is modeled in the following formula:

$$p(D = 1|\omega, c) = \frac{1}{1 + e^{-v_\omega . v_c}}$$

, where the model learns v_ω and v_c. The aim is to maximize the log probability of the pairs seen in the data and minimize that of pairs not seen. This can be written as:

$$arg\,max_{v_\omega , v_c} \left(\Sigma_{(\omega,c)\epsilon D} \log \sigma(v_\omega . v_c) + \Sigma_{(\omega,c)\epsilon D'} \log \sigma(-v_\omega . v_c) \right)$$

, where $\sigma(x) = \frac{1}{1+e^x}$ and $D'$is made of pairs (ω, c) which are randomly constructed and assumed to be incorrect. This part provides the model with what is referred to as negative sampling in the literature. The models learn the parameters by optimizing this objective and words appearing in similar contexts end up having similar embeddings.

1.3.2 Simple Description of the SkipGram Algorithm

To put it simply**,** SkipGram algorithm can be boiled down into four steps:

1- Take a neural network with three layers: input, hidden, and output
2- Feed the network with a word and train it to predict its neighboring word(s)
3- When done with the training, remove the output layer
4- Pick a word from the vocabulary, give it to the trained network; the output at the hidden layer is the embedding of the input word

One question might be which (ω, c) pairs the network is given. Here is a nice example of a training set given by Chris McCormick[2] where the window size is two. The window size is the number of neighboring words we want to consider for a word, and two means two words before and two words after the target word.

[2] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

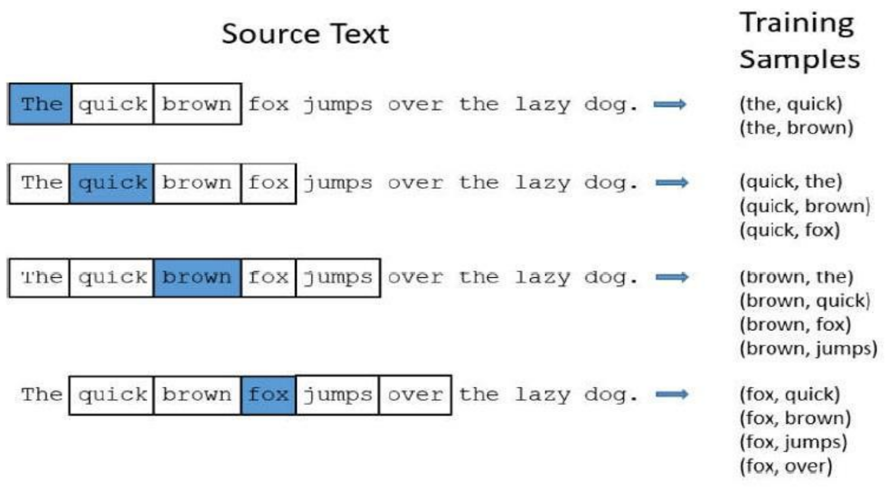


Figure 1.3. An example of a training sample for word2vec.

Another question could be how the network receives the input and makes the prediction. The answer is: since the neural networks only work with numbers, each word is fed to the net in the form of a one-hot vector. The size of this vector is the size of the vocabulary and the value of its all elements is zero except for one element which represents the word itself in the vocabulary. For instance, if you have a corpus with three words in the vocabulary, say, "*cat*", "*chase*", and "*dog*" (V=[cat, chase, dog]), the one hot vector for each of this word will be like this: *cat*= [1 0 0] ; *chase* = [0 1 0] ; *dog* = [0 0 1]

With regard to making a prediction, the network uses a softmax classifier to predict the neighboring words. The results of these predictions are compared to the ground truth values, the context vectors, which are also given to the network as outputs in the form of one-hot vectors. Based on the difference between the predictions and real values, the weights are adjusted. Figure 1.4 demonstrates the architecture of the model.

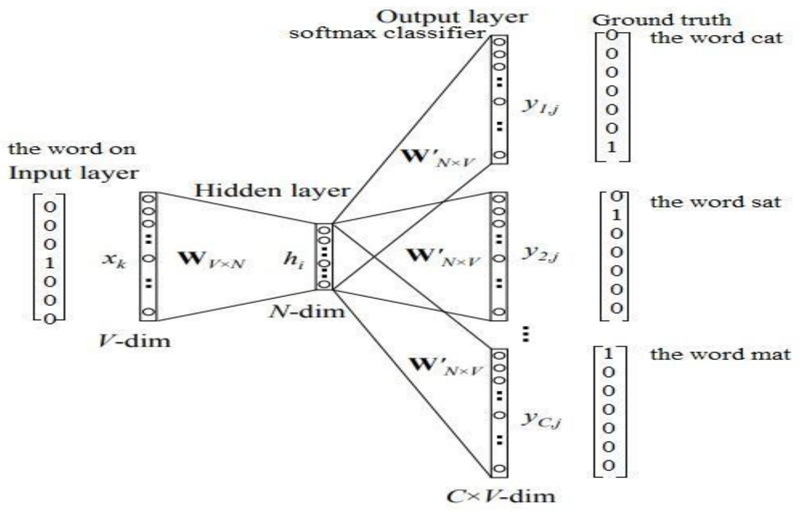


Figure 1.4. SkipGram model. The inputs and labels are given to the network in the form of one-hot vectors. The output layer uses a softmax classifier to make predictions. Then the predictions are compared against the ground truth to update the weights.

The final question is, after learning the weights and removing the output layer, how we can get the word embeddings. Figure 1.5 illustrates this clearly. Imagine the matrix in the figure is your hidden layer which contains the learned weights. In order to get the embedding for your intended word, you just need to multiply its one-hot vector by this matrix.

Weight matix of the hidden layer

one-hot vector of the intended word

Embedding of the intended word

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

Figure 1.5. How to get the embedding. (Adapted from Chris McCormick's article on word embedding[3])

As you can see the neural network learns the word embeddings based on what it receives in its input and output layers. For example, in the SkipGram model, the input is the target words we would like to learn the embeddings for and the output layer is what

[3] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

we define as the contexts of the words. Mikolov et al (2013) used just words themselves in input and output layers whereas other researchers suggest using words which are labeled with their dependency relations in a sentence in the output layer (Levy & Goldberg, 2014) or words and their character n-grams in the input and output layers (Bojanowski et al. ,2016).

1.4 Compositional Semantic Models

Compositional semantic models aim to build vectors for expressions longer than a word like phrases and sentences. Generally, these models create the vectors of phrases/sentences by composing the distributional representations of the words they contain. To do this, they adopt different approaches from defining the composition model as simple arithmetic operations (Mitchell & Lapata, 2008; 2010) to more sophisticated functional (Baroni & Zamparelli, 2010) and neural-based models (Socher et al., 2012).

In the simple approach, which is also our approach, the values in the input word vectors are summed, multiplied or concatenated to form the phrase/sentence vector. The logic behind these models is that if the values in the words vectors show the association of these words with the corresponding context, then the vector of the phrase that is composed of these words should inherit all these contextual features.

1.5 Research Questions, Aims, and Objectives

Each of the above-mentioned researchers, Mikolov et al (2013), Levy & Goldberg (2014) , and Bojanowski et al. (2016), claim different qualities for the embeddings resulting from their methods. Furthermore, all of them built their models with huge corpora (Google news or Wikipedia or a combination of both) and did not report any results on embeddings built on corpora of smaller size.

In this work, we first build three semantic spaces using two corpora of different sizes (British National Corpus with 100 million words, and Wikipedia with 600 million words) and three different models (BOW, syntax-based, and Fast-Text). We would like to know how different kinds of contexts and different sizes of corpus would affect the performance of resulting word embeddings on word similarity and relatedness tasks.

Moreover, we combine the resulting word embeddings to build sentence vectors. We use concatenation, summation/averaging and multiplication and evaluate the resulting sentence vectors on two (phrase/sentence) similarity tasks. Mitchell & Lapata (2008; 2010) used the same composition models and evaluated their models on the phrase similarity task. They utilized word vectors which were built using the traditional method (count-based) while we use word embeddings built through the predictive method (neural network based). We would like to know how our results might be different from theirs and which word embeddings and composition model best suit the paraphrase detection.

To sum up, the goal of this work is twofold: 1- To investigate the impact of different types of contexts on word embeddings produced by the SkipGram algorithm. 2- To experiment with the resulting word embeddings (from the first part) by combining them through different composition models (concatenation, summation/averaging and multiplication) to see which word embeddings and which composition model yield the best result for the task of paraphrase detection. Our research questions are:

For the word embeddings part,

1- How does the type of context affect the type of semantic information captured by the word embeddings built through the SkipGram algorithm?
2- Is the size of the corpus important? If so, which type of context (bag-of-words, syntactic, and character n-gram) needs the biggest corpus (and in turn, smallest corpus) to produce the embeddings of comparable quality?

And for the composition part,

3- For the task of paraphrase detection, which of the embeddings built in the first part is the most suitable for building sentence vectors?
4- For the task of paraphrase detection, which of the composition models (summation/averaging, concatenation and multiplication) is the most suitable for combining word vectors?

1.6 Thesis Structure

This thesis is structured as follows: In Chapter 2, we give a background on the concepts and tools we refer to in different parts of the thesis. Chapter 3 covers the literature review by discussing what has been done so far both for word and sentence representations. Chapter 4, titled Methodology, goes through the methods and principles we used to perform our experiments. Since we conducted two different sets of experiments (i.e. word and phrase/sentence), we divided the chapter into two parts where the first part deals with different ways of building word embedding and the second part concern the composition models. Chapter 5, Results and Discussion, presents the results achieved through evaluating word and phrase/sentence representations. Thus, this chapter also comes into two different parts where the first part offers the results related to the evaluation of word embeddings and the second part presents the results of evaluating composition models. Chapter 6 concludes the work by listing what we have learned during this study and what we are aiming to do in the future.

2 Theoretical Background

In this chapter, we explain some concepts and terminology that we referred to in the next following chapters.

2.1 Different Kinds of Semantic Relations

Word embeddings contain different types of semantic information. In order to know what kinds of information can be found in word embeddings, we first need to know what sorts of semantic relations exist between words. There are different semantic relations between words: Synonymy, antonymy, homonymy, metonymy, and etc.

Synonyms are words which have similar meanings. Synonyms are usually different from each other at least in one semantic feature. This feature sometimes is denotative which means that it comes from a real world difference like the difference between the words *walk, lumber, stroll, stride*, etc. On the other hand, it is sometimes connotative and more related to the feelings of the speaker toward the referent rather than a real difference in the world; like the difference between the words *die, pass away, kick the bucket*, etc. There are not many absolute synonyms in a language. For example, *sofa* and *couch* are almost complete synonyms but they collocate with different words: we can say *couch potato* but not **sofa potato*.

Hyponymy is the relationship between a general term (hypernym) and a specific instance of it (hyponym). A hypernym is a word or phrase whose semantic field is broader than its hyponym. For example, the words *apple* and *orange* are hyponyms of the word *fruit* and co-hyponyms of each other.

Meronymy denotes a semantic relation between a thing and its constituent part. That is, X is a meronym of Y if Xs are parts of Y. For example, a *tire* is a meronym of *car*.

2.2 Distributional Semantic Models

Distributional Semantic Models (DSMs) are computational models to represent words. In these models, each word is represented by a vector of real numbers in a space where semantically related words are close and unrelated words are distant. Despite the variations in building these models, the theories and assumptions behind them are the same. All these models are based on the Distributional Hypothesis (Harris, 1954) which argues the meaning of a word can be induced from the contexts in which it appears. Historically, these models are categorized into two categories of count-based and predictive. There is a detailed description of the models in the Introduction chapter.

2.3 Compositional Semantic Models

Compositional semantic models are models that aim to build vectors for expressions longer than a word i.e. phrases and sentences. The lexical vector space models are based on the distributional hypothesis, which states “Words that occur in similar contexts tend to have similar meanings” (Turney & Pantel, 2010). But what is the intuition behind the vector spaces for larger expressions? Regarding the larger expressions, there are two main approaches: either, extending the distributional hypothesis from words to phrases (by phrase we mean every constituent more than one word up to a sentence) and arguing that phrases have similar meanings if they occur in similar contexts, or taking a totally opposite approach by claiming that the meanings of phrases are derived by the meaning of their parts and how those parts are combined. The former fits in the framework of the context-theoretic theory proposed by Clarke (2008) and the latter resonates with the proponents of compositionality (Werning et al., 2012). These approaches are discussed later in more detail.

2.4 Dependency Parsing

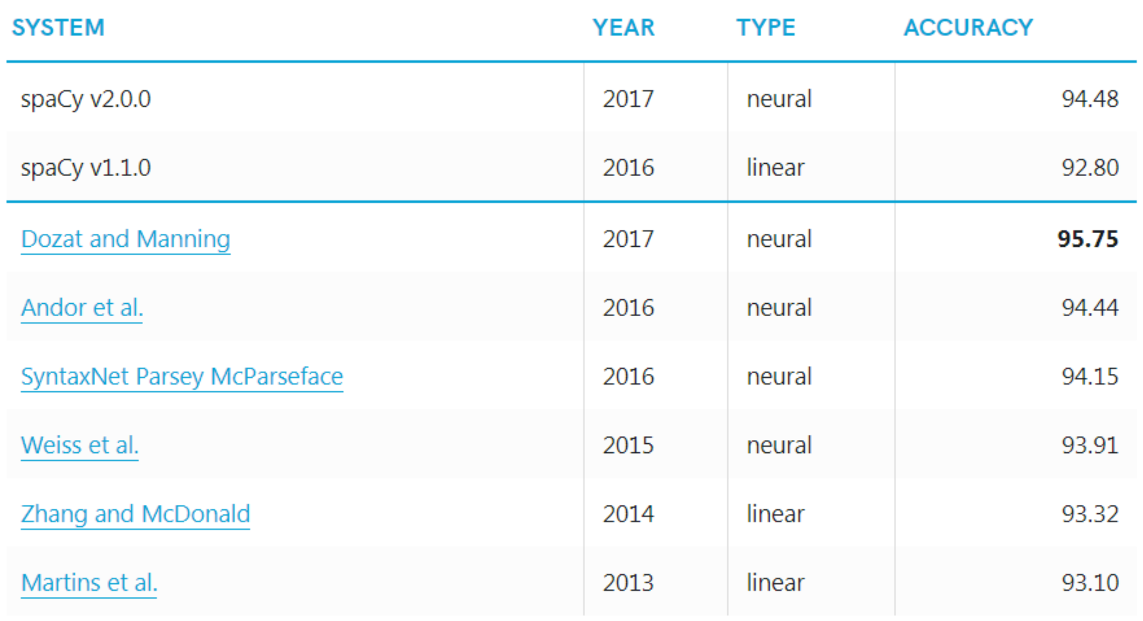
We define three types of contexts, bag-of-words, syntactic and character n-gram, to build three different semantic spaces. For our syntactic context, we had to parse the corpora we were working with. Parsing a sentence means assigning a syntactic structure to it. There are two main approaches to parsing a sentence: constituency parsing and dependency parsing. A constituency parse tree decomposes a sentence into sub-phrases. The terminals are the words in the sentence, the non-terminals in the tree are types of phrases, and the edges are unlabeled. On the other hand, a dependency parse tree connects words based on their relationships. Each vertex in the tree represents a word, child nodes are words that are dependent on the parent, and edges are labeled by the relationship. In this kind of parser, first the headword is extracted and its relations with other parts of the sentence are discovered.

These head-dependent relations give an approximation to the semantic relations between predicates and their arguments. For example, for the sentence "*Who wrote Harry Potter?*" the headword is *wrote* which is in relation with *who* and *Harry Potter* with the former being its nominal subject (nsubject) and the latter its direct object (obj). Having this information, it is not difficult for example for a question answering system to find out that the user is looking for the author of the book.

In conclusion, we can use a constituency or a dependency parser depending on our goal; the former gives us the sub-phrases in the sentence and the latter provides us with the dependency relations between words. Since we were looking for the dependency relations, we selected a dependency parser. There are many automatic dependency parsers in the field (Dozat & Manning, 2017; Andor et al, 2016) but we decided to choose the one which is offered by spaCy. In the following paragraph, we shortly describe this parser.

Spacy is an open-source library written in Cython (which is an extension of Python designed to give it a C like performance). It is quite fast and provides us with different tools for different tasks from tokenization to dependency parsing. spaCy's dependency parser's accuracy is within 1% of the best available (Choi et al., 2015).

Penn Treebank/Wall Street Journal (Marcus et al, 1993) is one of the popular datasets which is used by the researchers to evaluate their parsers. Figure 2.1 shows the accuracy scores achieved by different parsers including spaCy's on the dataset.



SYSTEM	YEAR	TYPE	ACCURACY
spaCy v2.0.0	2017	neural	94.48
spaCy v1.1.0	2016	linear	92.80
Dozat and Manning	2017	neural	**95.75**
Andor et al.	2016	neural	94.44
SyntaxNet Parsey McParseface	2016	neural	94.15
Weiss et al.	2015	neural	93.91
Zhang and McDonald	2014	linear	93.32
Martins et al.	2013	linear	93.10

Figure 2.1. spaCy's parser's accuracy compared to the other parsers in the field.

2.5 Softmax Classifier

The Softmax function takes an n-dimensional vector of real numbers and transforms it into a vector of real numbers in the range of 0 to 1, which add up to 1.

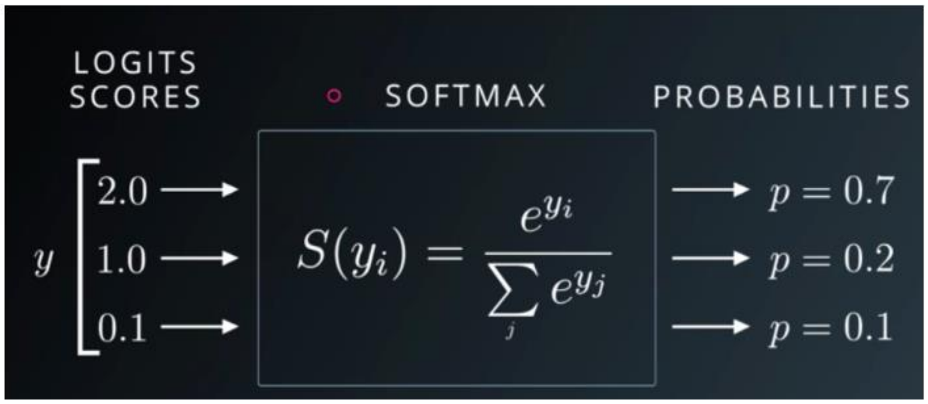


Figure 2.2 Softmax function which takes a vector of real numbers and turns it into probabilities (taken from Udacity Deep Learning Slide on Softmax)

The function breaks the whole (1) with maximal element getting the largest portion of the distribution, but other smaller elements getting some of it as well. This property of

Softmax function that it outputs a probability distribution makes it suitable for probabilistic interpretation in classification tasks (see Figure 2.2).

2.6 Similarity vs. Relatedness

WordSim353 (Finkelstein et al., 2001.) is a dataset which is used to evaluate models which produce word vectors on the task of measuring similarity. This dataset contains 353 pairs of words which are annotated by human judges based on their level of similarity where the most similar words are given score 10 and the least similar ones 1. However, Agirre et al. (2010) argued that some pairs are more related than similar. For example, *computer* and *keyboard* are related, and *train* and *car* are similar. They divided the dataset into two sets, WordSim353-Similarity, and WordSim353-Relatedness, and classified synonyms, antonyms, identical, and hyponym-hyperonym pairs as similar with different degrees (score 1 to 10) of similarity. They grouped meronym-holonym and the rest of the pairs (none-of-the-above) as related with different degrees of relatedness (1 to 10).

We used the dataset to evaluate our word embedding models, therefore when we talk about similar words we mean words which are synonyms, antonyms, identical, and hyponym-hyperonym and we refer to words as related, we mean words which are meronym-holonym.

3 Literature Review

Sentences are made of words; therefore in order to make proper representations for sentences, we need to create appropriate representations for words. This chapter is divided into two parts. In the first part, we discuss different methods used by researchers to build word vectors and in the second part, we deal with the works related to phrase/sentence vectors. Finally, we talk about our own contribution.

3.1 Building Word Vectors

The Distributional Hypothesis (Harris, 1954), which argues that the meaning of a word can be derived from the contexts in which the word occurs, has proved successful in the field of natural language processing. As we explained in the introduction, there are two main approaches to building word vectors: count-based and predictive.

In both methods, the dimensions of each vector are the features (the word's context items) and the coordinates are a measure of the association between the word and the features. However, in count-based models, the features are explicitly determined by the researcher who is building the vectors, whereas, in the predictive models, the neural network itself selects a number of features to represent a word with. The number of features representing a word is the dimensionality of word vectors (embeddings)[4] that can be determined by the researcher.

Many researchers experimented with count-based models (Church & Hanks, 1990; Padó & Lapata, 2007; Bullinaria & Levy, 2007) with the aim of improving the quality of word vectors. Other researchers (Mikolov et al., 2013; Pennington et al., 2014; Levy & Goldberg, 2014) did the same with predictive models. In the following paragraphs, we discuss some of these attempts to see what researchers have done so far to build different word vectors using both count-based and predictive models.

[4] Word vectors are referred to as word embeddings in the predictive models

3.1.1 Count-based Approach

As mentioned earlier, these models represent a word in the form of a finite-dimensional vector of real numbers in a vector space. The dimensions of this vector are the word's context items (e.g. co-occurring words) and the coordinates are a measure of the association between the word and the context items (e.g. the co-occurrence count). As you can see, each vector is defined by two main parameters: the dimension and the coordinates. Hence, in order to build word vectors with different characteristics, the best thing to do is to tweak these two parameters. Researchers have tried different ways which are explained in the following paragraphs.

3.1.1.1 Weighting Schema

In order to improve the performance of our count-based models, we can use a weighting function to turn the counts into more informative values. For example, if our target word is "*shoes*", a good weighting function would intensify more informative words like "*wear*" and down-weight the less informative ones like "*these*". Pointwise mutual information (PMI) introduced by Church and Hanks (1990) can replace the counts. Another variation of this score is positive-PMI which has been used by other scholars (Bullinaria & Levy, 2007). Curran (2003) has presented a nice overview of various weighting methods in his dissertation.

3.1.1.2 Other Sorts of Contexts

If we use word-word co-occurrences as contexts to create a semantic space, we will end up having a space where the closeness of word vectors implies semantic relatedness or domain similarity (Turney, 2012). However, the semantic space can be built using other types of contexts like the word's syntactic contexts. These contexts contain the words that are related to the target word by a syntactic dependency relation in a sentence (Padó & Lapata, 2007). The features extracted from this context make a space in which closeness of vectors suggests functional similarity (Turney, 2012)

3.1.1.3 Dimensionality Reduction

As mentioned earlier the rows of the matrix, built through counting the co-occurrences of the words, are word vectors. These vectors are very high-dimensional since the number of their columns equals the size of vocabulary (V) in the corpus. In fact, there are V^2 entries in the matrix which are estimated from the corpus. Considering the huge number of entries, this estimation is poor. Furthermore, there are some dimensions which are not informative enough and do not provide useful information for distinguishing the words. Moreover, some of these entries do not add extra information, for example, similar words which are used as column features will represent similar evidence. For these reasons, it is important to modify the matrix in a way that the remaining entries provide informative and effective information. To this end, some matrix mathematical operations are done on the matrix to reduce its dimensionality so that each feature is informative enough.

There are different ways of conducting these operations and singular value decomposition (SVD) is the most widely used one. SVD factorizes a matrix (M) into the products of three matrices $U\Sigma V^T$ where U and V are in column orthonormal form (the columns are orthogonal and have unit length, i.e., $UU^T=I$) and Σ is diagonal matrix of singular values. The matrices M and Σ are of the same rank, thus if the matrix M is of rank r, the Σ is of rank r too. The lower dimensional projection acquired through SVD has shown the ability to identify the latent meaning from the rows and columns of the original matrix (Landauer & Dumais, 1997) by reducing the noise in input matrix and discovering higher order co-occurrence among the rows of the matrix (Lemaire & Denhiere, 2008).

Other methods of dimensionality reduction include principal component analysis (PCA), canonical correlation analysis (CCA), non-negative matrix factorization (NMF), etc. Each of these methods has its own pros and cons compared to SVD. This thesis does not concern the count-based method and as a result, does not deal with dimensionality reduction methods, the explanation of such methods is not considered. However, the interested readers are referred to Koren et al. (2009)'s work which provides a comprehensive survey on this matter.

Latent Dirichlet Allocation (LDA) is another traditional method used to build word vectors. Since our main focus in this thesis is predictive models, especially SkipGram algorithm, we do not explain LDA, but interested readers are referred to Latent Dirichlet Allocation paper by Blei et al. (2003) for comprehensive information.

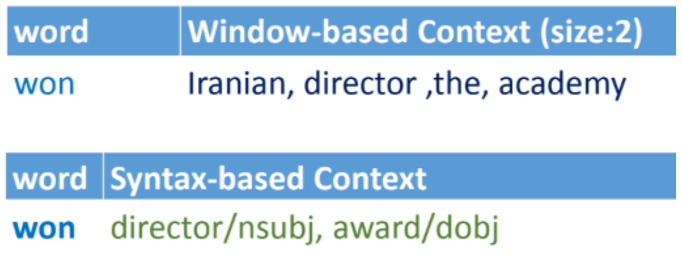
3.1.2 Predictive Approach

In predictive methods, a word's embedding is built by predicting the word in the context it occurs regardless of the order of the words in the context (Mikolov et al., 2013; Pennington et al., 2014). These models consider the words occurring both to the left and right of a target. Word embeddings trained through such models have shown to represent the semantics of the word better than the embeddings created using language modeling where only preceding words are considered as the context of the word (Faruqui and Dyer, 2015).

3.1.2.1 Context in word embedding with the predictive approach

In the Skip-Gram model proposed by Mikolov et al. (2013) the context of a word, ω, are the words around it. Moreover, the context vocabulary, C, is the same as word vocabulary, W. However, this limitation is not obligatory, and the context set can be bigger than the word set.

Levy & Goldberg (2014) suggested a different kind of context by parsing the corpus with a dependency parser and exploiting the syntactic information as extra contextual information. In order to understand their model, let's take this example: *"Iranian director won the academy award."* Imagine we want to feed the algorithm with (word, context) pairs using two different types of contexts. As you can see in Figure 3.1, for the word "*won*", the context in syntax-based (DEPS) approach is more focused by filtering out the co-incidental words (e.g. *Iranian*) and taking into account the more related word (*award*) even though it is in a long distance from the target word (*won*).



word	Window-based Context (size:2)
won	Iranian, director ,the, academy

word	Syntax-based Context
won	director/nsubj, award/dobj

Figure 3.1. Contexts for the word “won” in window-based (BOW) and syntax-based approaches.

To build word embeddings, two methods have been described so far: BOW and DEPS. These methods are similar in the sense that both rely on the neighboring words of a target word to create the embeddings of the word. However, they are different in a way that the former does not take the syntactic role into account but the latter does. These two algorithms provide us with good quality word embeddings, but they fail to create embeddings for rare words or the words they have not seen during the training. To tackle the problem another algorithm was suggested by researchers at Facebook which is called FastText.

Bojanowski et al. (2016) suggest a new definition for word/context. Their model is different from other embedding models in the way it treats each word. In particular, other models consider every single word as the smallest unit that we look for its representation, whereas FastText treats a word as a combination of n-grams of characters. For example, the word *funny* is composed of [*fun, funn, funny*], [*funny, unny, nny*] etc, where n is of length one up to the actual length of the word. This way of treating words have some benefits over the models which assume the word itself to be the smallest unit.

One of the advantages of this model is that it can find vector representations for rare words. This quality is due to the fact that these words are broken into character n-grams, they can share n-grams with words which are more common. The other advantage is that we can find vector representations for words which do not exist in the dictionary because these words also are broken into character n-grams. It is obvious that other word-based methods fail to do so. For example, if the word is *fanabsolutastic*, which probably does not exist in any corpus, Word2Vec algorithm might return either a zero vector or a random one, while FastText might produce vectors better than random vectors and come up with a vector which is similar to fantastic.

3.1.3 Other Models

All models discussed so far focus on neighboring words to create the embeddings; they combine the notion of semantic similarity and conceptual relations. In other words, they fail to tell the synonyms and antonyms apart and place them close together in the semantic space. For example, words like east and west or cheap and expensive appear in the similar contexts therefore, the models learn similar embeddings for them.

To tackle the problem, some researchers (van der Plas & Tiedemann, 2006) suggest other kinds of contexts to build embeddings. They argue that translations of a word into other languages which can be found in parallel corpora are a good context to learn word embeddings that can make a distinction between synonyms, and antonyms, (co)hyponyms and other related words. This argument is justified since the word apple is usually translated neither to fruit nor pear; neither is good translated into bad. They compared their proposed system with syntax-based models and achieved higher precision and recall scores for the task of synonym extraction. Despite our huge interest to investigate their proposed model, we could not experiment with that due to the time constraints. However, it is at the top of our future work after finishing the current study.

3.1.4 Evaluating the Word Embeddings

Using qualitative and quantitative analyses, Levy & Goldberg (2014) demonstrated that word embeddings built through syntactic context (DEPS) represent functional similarity or co-hyponymy relation. For example, for the word Florida (a state in the U.S) bag-of-words contexts produce meronyms (counties or cities within Florida) like Gainesville and Tallahassee, while dependency-based contexts generate co-hyponyms (other US states) like Louisiana and California.

Melamud et al. (2016) evaluated the performance of word embeddings on different intrinsic and extrinsic tasks. They changed the context type and dimensionality for the embeddings they built to find out how the performance changes consequently. They used window-based (BOW), syntax-based (DEPS) and lesser-known substitute words approach (Yatbaz et al., 2012) (SUB) as different context types and 25, 50, 100, 200, 300, and 600 as different dimensions.

The intrinsic benchmarks used for evaluation include WordSim353 similarity and relatedness, Simlex-999 in which they calculated the Spearman's correlation between the rankings of the word pairs induced from human annotations and embeddings and also the accuracy of the system for TOEFL task. As for the extrinsic tasks, they put the embeddings to the test by evaluating their performances on Dependency Parsing (PARSE), Name Entity Recognition (NER), Coreference Resolution (COREF) and Sentiment Analysis (SENTI).

They concluded that the performance on the intrinsic tasks increases as the number of dimensions does and it reaches to its optimal point around 300 dimensions for all types of contexts.

As for the context type, there are noticeable differences in results with the most obvious one being the WordSim353 relatedness task in which the BOW context with the largest window size performs best as expected since it covers topical similarity. On the other hand, DEPS and SUB embeddings achieve better results in WordSim353 similarity and Simlex-999 datasets. The performance of all sorts of contexts is comparable in TOEFL task.

As for the extrinsic tasks, surprisingly for some tasks like NER, the optimal performance can be reached with as few as 50 dimensions. Moreover, increasing dimensionality even deteriorates the performance. They relate this behavior to the size of training data used by the classifier and emphasize the importance of tuning the dimensionality of the embeddings based on the task at hand.

They found out that the embeddings which work well on functional similarity and badly on topical similarity (DEPS, SUB, and W1) work best in PARSE. For the rest of the tasks, they couldn't find a specific context that outperformed the others.

In the next step, they experimented with concatenating the embeddings. They reported that in order to make the most of concatenation, we should first reach an optimal performance with the dimension and then concatenate these optimum dimensions. They also stated that the concatenation of window-based method with the window size of 10 with one of the functional contexts, SUB, DEPS or BOW1, yields the best results.

The takeaway lessons from Melamud et al (2016) are:

1. The dimension of embeddings matters with 300 usually being the optimum (but not for every task). However, depending on the available data for training sometimes lower dimensions work better.
2. You can get better results by concatenating embeddings resulting from different contexts to get better results on condition that each embedding has set to its optimum dimension when evaluated on the task individually.

Bojanowski et al. (2016) evaluated FastText with corpora of different sizes. They found that FastText is more robust to the size of the corpus compared to the BOW model. They also assessed the model on various word similarity datasets of different languages and concluded that FastText outperforms Word2vec algorithms (CBOW, SkipGram) in morphologically rich languages. They did not observe any noticeable difference in the performance of the algorithms on English datasets.

3.2 Building Phrase/Sentence Vectors

When it comes to building phrase vectors, the first thing that comes to mind is to combine the vectors of the constituent words through some arithmetic operations. This approach has been explored vastly in a series of seminal papers by Mitchell & Lapata (2008; 2010).

In the simple additive model suggested by (Foltz et al., 1998; Kintsch, 2001; Landauer & Dumais, 1997), the values in input word vectors are summed to form the phrase vector. The logic behind such a model is that if the values in the words vectors show the association of these words with the corresponding context, then the vector of the phrase that is composed of these words should inherit all these contextual features.

As an alternative to the additive model, Mitchel & Lapata (2008; 2010) propose a "component-wise multiplication" model in which input vectors are multiplied component by component. This way those contextual elements which are not associated with all input words vanish and shared elements are heightened. This model is in line with the intersection view in formal semantics.

Some other methods have also been proposed which more or less can be regarded as weighted variations of additive models. For example, Mitchell & Lapata (2008; 2010) proposed a weighted additive model in which each input vector is multiplied by a fixed weight. For example, in case of combining subject-verb vector, the subject vector might be multiplied by 0.2 and the verb vector by 0.8, with the intuition that the verb has more weight in depicting the sentence meaning rather than the subject. Other summative weighted models that are worth mentioning are Guevara (2010) and Zanzotto et al. (2010) where each element is output vector is a weighted sum of all elements of input vectors. But the problem with such simplistic methods is that they fail to capture important aspects of compositionality like word order, ambiguity, grammatical relations and etc.

A different series of models inspired by formal semantics (Montague, 1970) which is the framework of models proposed by Coecke et al. (2010) and Grefenstette et al. (2011) look at the composition from the functional approach.

In functional approaches, not all words have standard distributional vectors but instead some words are in the form of arguments which are represented by standard vectors and some other words are functors that apply some functions on these arguments. For example, in additive models, the vector for *dogs bark* is created by adding the distributional vectors of *dogs* and *bark*, whereas in functional approach *bark* is a function that operates on *dogs*.

Due to the mathematical considerations, functions that operate on the arguments are limited to the class of linear transformation. They are ordered set of weights that can be seen as vectors and be fed to other functions as input or output. This property of functions allows mapping within and across vector spaces.

Following the functional approach, Baroni and Zamparelli (2010) define adjectives as functions and nouns as arguments, each represented by matrices and vectors respectively where adjective-noun compositions are obtained by multiplying the adjective matrices by the noun vectors. The matrices for adjectives are learned from the corpus by collecting adjective-noun pairs and through supervised learning. But the question is whether the same method can be used for linguistic units larger than adjective-noun?

Socher et al. (2012) introduced a recursive neural network model that learns compositional vector representations for phrases and sentences of arbitrary syntactic type and length. Their model associates each node in a phrase tree with a vector and a matrix where the vector contains the meaning of the node and the matrix shows how the node changes the meaning of neighboring words or phrases. According to Baroni et al. (2013) "This results in a very rich and flexible representation of phrases, although the linguistic intuitions behind it are not clear."

Le & Mikolov (2014) proposed a paragraph vector to learn representations of sentences, paragraphs, and documents based on SkipGram model and by adding an explicit paragraph feature to the input of the neural network. They showed better performance compared to the bag of words methods.

Other neural networks based approaches towards meaning representation of sentences called sentence embeddings include Conneau et al. (2017) where they trained a universal sentence representations using the supervised data of the Stanford Natural Language Inference (SNLI) datasets which outperformed unsupervised methods like Skip Thought vectors (Kiros et al., 2015) on a wide range of transfer tasks.

Hill et al. (2016) presented a systematic comparison of different methods of learning distributed sentence representations and showed noticeable variation in their performance and concluded that the performance of the model depends on the settings the representation is going to be applied.

To sum up, the approaches towards vector representation for sentences can generally be categorized into three main classes: first: the sentence vectors are created by simple arithmetic operations on word vectors, the methods that mainly explored by Mitchell & Lapata (2008; 2010).

Second: Functional approach in which some words are functors and the rest are arguments, and the representation of phrases is the production of functors acting on arguments. These methods have been investigated by many researchers including Coecke (2010), Baroni & Zamparelli (2010), Sadrzadeh & Grefenstette (2011).

Third: The neural-based approaches towards the meaning representation of sentences or so-called sentence embeddings examined by Le & Mikolov (2014), Conneau (2017), Hill (2016) and so many others.

3.2.1 Evaluating the Compositional Models

The composition models are usually evaluated by the existing datasets in the field such as the one introduced by Mitchell & Lapata (M&L) (2010). The dataset is made of pairs of two-word phrases which were judged by 204 native speakers of English of different age and gender and assigned a number on a scale 0 to 7 where 0 represents none and 7 complete similarity between the pairs.

Mitchell & Lapata (2010) presented a framework for vector-based semantic composition and formulated composition as addition and multiplication and used the M&L dataset to evaluate their models. They observed that different combinations of composition models and word vectors yield different results. Specifically, the additive model performs best if it uses the word vectors resulting from Latent Dirichlet Allocation (LDA) and the multiplication works best with the word vectors resulting from simple co-occurrence model.

One of the other datasets to evaluate compositional models is Microsoft research paraphrase corpus (MSR) (Dolan et al., 2005). Milajevs et al. (2014) investigated the addition and point-wise multiplication compositional models, to compare the performance of count-based versus predictive word vectors. They evaluated their models on MSR dataset and concluded that the combination of additive compositional model and predictive word embeddings (BOW) perform best.

According to Wieting et al. (2015) even though complicated methods for building sentence embeddings (e.g. LSTM) work strongly on the specific task they have been trained for, a simple method like representing a sentence's vector by the average of its word vectors outperforms these complicated models in some other tasks like sentence similarity and entailment. In other words, while the neural-based models perform well in in-domain tasks, they fall short in out-of-domain tasks. On the other hand, word-averaging shows a robust performance by functioning well both in in-domain and out-of-domain tasks and even beating the complicated method by a noticeable margin.

3.3 Contributions

Researchers have evaluated different models of word embeddings (bag-of-words, syntactic, and character n-gram) and reported the results; However, most of these evaluations (Levy & Godberg, 2014; Melamud et al., 2016) were done using the embeddings which were built with huge corpora (10B words in Melamud et al. (2016)). To the best of our knowledge, no one has ever tried to compare the performance of these three predictive models with corpora of smaller size. Therefore, we would like to see whether the similar results are achieved when the size of the corpus is quite small.

Moreover, no one has ever tried to combine the word embeddings resulting from the syntactic context and character n-gram to build sentence vectors in the paraphrase-detection task. We use simple composition models (concatenation, summation/averaging, and multiplication) to build our sentence vectors. We would like to find the best combination of composition model and word embeddings for building sentence vectors.

Furthermore, Mitchell & Lapata (2010) used the same composition models but with the word vectors resulting from the traditional method. We use these composition models with neural-based (predictive) word embeddings to see how our results are different from theirs.

From now on, the bag-of-words, syntactic, and character n-gram are also referred to as BOW, DEPS, and FastText respectively.

4 Methodology

This chapter is divided into two parts: In the first part, we explain the methodology we used to build our word embeddings and in the second part we deal with the composition models.

4.1 Methodology- Word Vectors

Apart from the main goal of this work, which is to investigate the effect of the type of context and the size of the corpus on the word embeddings, we aimed to show that everybody can build these embeddings and benefit from them regardless of their computational resources and computer science background. In other words, we would like to show that the use of neural-based word embeddings is not restricted to people who have access to GPUs or have extensive knowledge in deep learning and computer programming. Thanks to different tools developed by the programmers in the field of natural language processing and free corpora of different sizes, everybody can make their own word embeddings with acceptable quality and use them in their research. By acceptable quality, we mean acceptable accuracy and F-score for tasks that need semantic judgment at word and sentence level.

With this in mind, we did all the computations, from dependency parsing of the corpora to building window-based (BOW, FastText) and syntax-based (DEPS) models on a core-i5 laptop with 8 Gigabytes internal memory (RAM). We also utilized some easy-to-use Python libraries including Gensim, spaCy, NLTK, and pntl (SENNA). In the following, we give a full description of the whole procedure of building word embeddings.

4.1.1 Corpora

Many of the neural-based models have used huge corpora to build the word embeddings, for example Melamud et al. (2016) benefitted from a corpus with approximately 10 billion words by combining English Wikipedia 2015, UMBC web corpus (Han et al., 2013), and English Gigaword newswire corpus (Parker et al., 2011). However, we argue that we still can build good quality word embeddings using much smaller corpora.

Sticking to the idea that all the computations had to be done on the laptop described before, we selected two available free corpora to investigate the effect of the size of corpus on the resulting word embeddings. These corpora include British National Corpus, BNC (Leech et al., 1994) with 100 million words, and Wikicorpus (Reese et al., 2010) which is 6 times bigger than BNC with 600 million words. We also built the embeddings using a very small corpus, Brown (Francis & Kučera, 1964), with 1 million words), but the results are poor which confirms our hypothesis that the size of the corpus affects the quality of word embeddings.

BNC (Leech et al., 1994) is a collection of samples of written and spoken English from a broad source with 100 million words. The corpus represents a cross-section of British English in the late 20th century whose last version, BNC XML Edition, was released in 2007. The written part of the corpus comprises 90% of it whereas the spoken part accounts for the remaining 10%. The samples for the written part are taken from regional and national newspapers, journals for different ranges of ages and interests, academic books, school and university essays, and many other sorts of texts. The spoken part includes samples of informal conversations which were recorded by volunteers from different age group, region, and social classes.

Wikicorpus (Reese et al., 2010) is a trilingual corpus of English, Spanish, and Catalan which is based on 2006 dump of Wikipedia and enriched with linguistic information. The corpus is annotated with lemma and part of speech information and is also sense-annotated using UKB word sense disambiguation algorithm. However, we did not benefit from any of these annotations and just utilized the raw-text for English part which is a very clean text (was cleaned and pre-processed by the providers) with 600 million words. The reason for selecting this corpus was its size. This corpus is big enough, compare to BNC, to help us analyze the impact of corpus size on the quality of word embeddings and small enough to run everything our laptop.

4.1.2 Tools

Having decided on the corpora, we had to choose one of the available tools (e.g. python libraries) to build the embeddings. We singled out Gensim which is created and maintained by Řehůřek et al.(2010) .

Gensim is a free Python library designed to extract efficient and convenient semantic information from documents. It can process unstructured raw data. Many algorithms have been implemented in Gensim such as Word2Vec (Mikolov et al, 2013), FastText (Bojanowski et al, 2016), Latent Semantic Analysis (LSA), and etc. These algorithms discover semantic information in the documents and are unsupervised which means they need no human intervention. We build our embeddings using SkipGram implementation of Word2Vec in Gensim.

There are several features that make Gensim an ideal tool to build word embeddings with. However, the most important one from our point of view is that Genism is memory independent. In other words, there is no need to load the whole corpus in RAM. This feature makes it possible to work with very large corpora despite having a limited RAM capacity.

4.1.3 Word Embedding Using the SkipGram Algorithm in Gensim

Gensim's Word2Vec algorithm expects an iterable of sentences as its input. Each sentence is a list of words. The command to build embeddings using Word2Vec includes several arguments that we will explain below:

```
model=Word2Vec(sentences, size, window, min_count, workers)
```

The first argument, `sentences`, needs to be an iterable of sentences where each sentence is a list of words (utf-8). For example, if we have a corpus of two sentences like "*First sentence*" and "*Second sentence*", the input needs to be in this form: `sentences` = [['first', 'sentence'], ['second', 'sentence']]. Although keeping the input as a Python list in the memory is very convenient, it consumes a lot of RAM and in case of a big corpus, we will run out of memory. In order to avoid this problem, we defined a function in which we read the corpus, sentence by sentence, and used yield command to return a list of each. Since yield returns a generator, the lists are created on the fly and do not stay in RAM.

Other parameters of the Word2Vec model are as follows:

`size`

It determines the size of the dense vectors we want to represents the words with (in other words the neighboring words or the context). You can specify different sizes and if the try up to the size of 600. Melamud et al. (2016) did a study using different sizes of embedding from 50 to 600 and found out that 300 is the optimum size across different tasks. Following their finding and considering the size of our corpora we set the size to 300.

`window`

It is the maximum distance between the target word and its neighboring words. This distance is calculated from the left and right side of the target word. If the position of a word is bigger than the window size, the word will not be considered as the context of the target word. For example, for the sentence "I know that my dog likes chasing cats." The context words within the window size of 2 for the word *likes* are *my, dog, chasing,* and *cats*. The remaining words will not be included despite being part of the same sentence.

`min_count`

This is the cut-off frequency. In other words, the model discards the words which do not fulfill the min_count. For example, if the `min_count` is set to 100, every word which occurs fewer than 100 times will be ignored by the model. It means that the model just builds the embeddings for the target words which occur more than 100 times in the corpus. This setting really affects the amount of memory used by the system or storage capacity to store the resulting model.

`workers`

This parameter defines the number of workers to use and really increases the speed when benefiting from GPUs. Since we built all the models using a CPU, we set this parameter to 1 in all experiments.

Now that we are familiar with different parts of the Word2Vec model in Gensim, let's see how we used the model to build the word embeddings.

The final note is that Gensim lets you build a FastText model using the same piece of code except that instead of Word2Vec, you should use FastText. Besides, FastText has three additional parameters :

```
- min_n: min length of char ngrams (Default 3)
- max_n: max length of char ngrams (Default 6)
- bucket: number of buckets used for hashing ngrams (Default
2000000)
```

We used the default setting.

4.1.3.1 Window-based word embedding with Brown, BNC, and Wikicorpus

Since we already had the NLTK library installed on our system, we had Brown corpus in its corresponding folder. However, we wrote a short script to modify the corpus so that we could work with it more conveniently. We created a single file out of the corpus where each line contains one sentence. Later on, we removed the punctuations and build the models on a punctuation-free corpus.

In order to build a BOW model we used the following piece of code:

```
model=Word2Vec(sentences, size=300, window=2,min_count=100,
workers=1,sg=1,hashfxn=hash)
```

The window size is set to 2 because we wanted to build the BOW2 model. For BOW5 and BOW10, this parameter is set to 5 and 10 respectively. The model's parameters have been already explained in the previous section except for `sg`, and `hashfxn`. As you remember from the literature review, the Word2Vec algorithm proposed by Mikolov et al. (2013) has two flavors: CBOW and SkipGram. By setting the *`sg`* value to 1, we tell the algorithm that we want it to use the SkipGram model. As for `hashfxn`, we set the value to hash to make sure that the same model is built by Gensim for different runs on the same data.

Having created BOW2 model with Brown, we carried on by building the models using the other corpora. We downloaded the BNC corpus from http://ota.ox.ac.uk/desc/2554. The corpus is in XML format, while we needed it to be an iterable. Fortunately, NLTK provides a package to read the corpora and we used BNC Corpus Reader to read the BNC corpus in our desired form.

```
from nltk.corpus.reader.bnc import BNCCorpusReader

a = nltk.corpus.BNCCorpusReader(root='D:\Thesis\BNC\Texts',
fileids=r'[A-K]/\w*/\w*\.xml')

sentences = a.sents()
```

After building the model with BNC, we continued taking the same steps to build the same model but this time using Wikicorpus, which was 6 times bigger than BNC. Hence, at the end of this step, we had built 3 BOW2 models with three corpora of different size. The reason to do so was to study the effect of the size of the corpus on the quality of resulting embeddings.

In the next step, we built two more BOW models, BOW5 and BOW10 with the Wikicorpus. The aim here was to verify the findings of other researchers' (Levy & Goldberg,2014; Melamud et al., 2016) who argue that the smaller window-size will lead to more functional similarity while the larger one produces embeddings which reflect more topical similarity. Therefore, we kept the size of the corpus fixed (we only used Wikicorpus) and changed the size of the window (5 and 10) to see how it affected the quality of resulting word embeddings.

The Wikicorpus we used was already pre-processed and cleaned by the providers, therefore we did not do any pre-processing, neither did we lower-case the words. In other words, we kept the text intact. Some researchers prefer to lower-case the corpus but we believe that it is not needed since the corpus is big enough to build quality word embeddings. Besides, since the models differentiate between a word in capital or in lower-case, it builds two separate embeddings for apple (a kind of fruit) and Apple (the company).

FastText embeddings were built in the same fashion.

4.1.3.2 Syntax-based word embeddings with BNC and Wikicorpus

After building the BOW and FastText embeddings, we started creating the syntax-based ones. To build these embeddings, we had to first parse the corpora with a proper dependency parser. Among all dependency parsers in the field, we decided to go with the one provided by spaCy. As explained in the first chapter the parser has a good level of accuracy compared to the other parsers, besides it is fast and easy to use.

Then we had to process the corpora to make it a suitable input for Gensim. By suitable we mean that Gensim could receive one target-context pair at a time. We wrote a piece of code to make sure that after parsing each sentence, each target word was paired with all the context words it was in dependency-relation with; and the context words were labeled in the exact same way they were labeled in Levy & Goldberg's (2014) proposed method. To check the correctness of the written code, we purposefully tried the same sentence used by Levy & Goldberg (2014) to see if the output of the code (which is the input of the Gensim Word2Vec algorithm) was what it was supposed to be. Hence, we gave the code this sentence: "*Australian scientist discovers star with telescope*". The output is what you can see in Table 4.1.

Word	Contexts
Australian	scientist/amod-1
scientist	Australian/amod, discovers/nsubj-1
discovers	scientist/nsubj,star/dobj,telescope/prep_with
star	discovers/ dobj-1
telescope	discovers/ prep_with-1

Table 4.1. The target, context pairs produced by the parser to be fed to Gensim Word2Vec.

As you can see, the (target, context) pairs are exactly the same as what Levy & Goldberg (2014) extracted for their model. For example for a target word, w , which has modifiers m_1,…, m_k and a head word h, the following contexts are extracted: (m_1,lb_1), …, (m_1,lb_k), (m_1,lb_{-1}), where lb is the type of dependency relation between the head and the modifier (e.g nsubj) and lb_{-1} is used to signify the inverse relation. It is also clear that relations that include a preposition are "collapsed" prior to context extraction, by directly connecting the head and the object of the preposition, and subsuming the preposition itself.

After making sure that the code provided us with the intended input for Gensim, we started to parse the Brown corpus. It took 30 minutes to parse the corpus and the result was saved in a file. Each line of the file contains two words with the first being the target and the second the context. For example, if the corpus was just one sentence like the sentence of Table 4.1, the first line of the output file would be *"Australian, scientists/amod-1"*. This was exactly we what we wanted as our input to the Gensim Word2Vec model. The next step was to build the model. We used the following code to build our syntax-based embeddings:

```
model=Word2Vec(sentences,size=300,window=1,min_count=100,
workers=1,sg=1,hashfxn=hash)
```

We treated each line of the input file as a two-word sentence and since the window size was set to 1, each word was paired with all its syntactic contexts one by one.

The next corpus to be prepared was BNC. The whole corpus was parsed after 24 hours and the result was saved in a file. To build the same model with Wikicorpus, we followed the same steps it took 6 days for the corpus to be parsed, which was what we had estimated given that the corpus was 6 times bigger than BNC. After having the parsed corpus, we used the same code to build the embeddings.

4.1.3.3 Semantic-based word embeddings

The last model that we were aiming to build and evaluate was the semantic-based model. The model suggests the same algorithm which is used in the syntactic-based model except that, this time, context words are labeled with their semantic roles in the sentence rather than their syntactic roles. We argue that this kind of context represents meaning at a deeper level than a simple syntactic structure. For example, consider the following sentences:

1- Iranian director won the academy award.

2- The academy award was won by Iranian director.

The syntactic and semantic contexts for the word *won* are described in Table 4.2:

Sentence Number	Syntax-based context	Semantic-based context
1	…, award/dobj,...	…, award/A1, ...
2	…, award/nsubjpass, ...	…, award/A1, ...

Table 4.2. In syntax-based context, the word *award*, as the context of the word *won*, is labeled differently depending on the voice of the sentence; whereas, it is labeled the same in semantic-based context.

As you can see in Table 4.2, the semantic-based context is not affected by the voice of the sentence and award is labeled as A1 (argument 1) of the verb won regardless of the sentence being active or passive.

Hoping to build word embeddings which are less susceptible to the surface structure of the sentence, we started to first label the corpora at hand with a proper semantic role labeler and then extract target-context pairs and build the models. After a thorough search in the field, we selected SENNA (Collobert, 2011)which is a fast and reliable automatic semantic role labeler. In order to access SENNA, we utilized nlpnet which is a free python library.

In order to extract (word, context) pairs, we first used SRL to extract verbs and their arguments. Then we discarded all stop words (in NLTK) from the arguments and then paired each verb with its arguments. Each argument was labeled with its semantic role. If an argument contained more than 2 words, only the first two words were paired with the verb individually and the rest were discarded. For example for the sentence: "*The disease infected both her family and friends*." you can see the target-context pairs in Table 4.3.

Sentence: The disease infected both her family and friends.	
Word	context
Infected	disease/A0, family/A1, friends/A1
disease	Infected/ A0-1
family	Infected/A1-1
friends	Infected/A1-1

Table 4.3. Target-context pairs for semantic-based context for the sentence: "The disease infected both her family and friends."

As you can see from the Table 4.3, verbs are emphasized which is, in fact, the essence of SRL whose task is to discover the relations between a verb and other parts of the sentence. With this method, words which are occurring with the same verbs are pushed close together and so are the verbs which are appearing with the same words.

Similar to the usual routine, we started with the Brown corpus and it took 12 hours to label the corpus and extract the target-context pairs. Then we carried on with the other corpora, beginning with BNC, and we noticed that we need at least 50 days (100*12 hours) to perform the same procedure with the BNC corpus let alone the Wikicorpus. We realized that we would not be able to finish the process within the time limits for this thesis. Therefore, we did not continue with this model.

4.2 Methodology-Phrase/Sentence Vectors

Compositional models aim to represent the meaning of a sequence of words by a vector which is built by combining the vectors of the words within the sequence. There are various ways of combining word vectors which are classified into three main approaches, as discussed in the literature review: simple models which combine word vectors regardless of their order or their syntactic or semantic relations, models which take linear word order into account, and models which incorporate complex grammatical structure.

The first series of models utilize simple arithmetic operations to blend word vectors; for example, Mitchell & Lapata (2008) used vector addition and point-wise multiplication by ignoring word order. Such a method cannot make a distinction between the sentences "*John loves Mary*" and "*Mary loves John*". Other approaches which are neural-network based and consider the grammatical structure of sentences (Socher et al., 2012, Conneau et al. 2017). In this work, we did not consider the neural-based approach because they are task-specific (Wieting et al., 2015) while we were aiming for a more general approach toward compositionality, as mentioned in the introduction.

We experimented with three different simple ways of combining word vectors which include: summation/averaging, element-wise multiplication and concatenation. The following paragraphs describe the process in more details.

4.2.1 General Framework

We applied three different operations to build our compositional models. We also used two different sets of word embeddings: 1- off-the-shelf embeddings (BOWs and DEPS provided by Levy & Goldberg, 2014, and FastText by Bojanowski et al., 2016), 2- the embeddings we trained on Wikicorpus using different contexts. In the following sections, we explain our methodology in more detail.

4.2.2 Composition Models

For a sequence of words $w_1w_2w_3\dots w_n$, we define the vector as $s=w_1\therefore w_2\therefore\dots\therefore w_n$, where $\therefore$ is the operator we use to combine the constituent words vectors. Since our goal is to experiment with simple composition models to compare our results with those of Mitchell & Lapata (2010), we decided $\therefore$ to be summation (+), point-wise multiplication (●), and nothing, which is when we concatenate vectors. Here is an example of building the sequence vector given its constituent word vectors; Imagine the vector for the word *happy* is: [1 0 4 7] and the vector for the word *ending* is: [2 3 0 9]; The resulting vector for the phrase "*happy ending*" is [3 3 4 16] using summation and [0.75 0.75 1 4] using Averaging. The phrase's vector using point-wise multiplication is [2 0 0 63], and finally, concatenation gives us [1 0 4 7 2 3 0 9] as the vector of the phrase.

4.2.3 Evaluation Sets

We evaluated our compositional model at two different levels: phrase-level and sentence level. For the phrase-level evaluation we used Mitchell & Lapata's (M&L) 2010 dataset and for the sentence level, we utilized Microsoft Research Paraphrase Corpus (MSR). In the following paragraphs, we introduce these datasets briefly.

4.2.3.1 M&L

Mitchell & Lapata's 2010 dataset (M&L) contains three types of phrases: adjective-nouns, noun-nouns, and verb-objects. The phrases are taken from BNC and are frequent to avoid any confusion from unfamiliar constructions. Each line of the dataset contains a pair of two-word phrases. The pairs were judged by 204 native speakers of English of different age and gender and assigned a number on a scale of 0 to 7 where 0 represents none and 7 complete similarity between the pairs.

4.2.3.2 MSR

Microsoft research paraphrase corpus (MSR) (Dolan et al., 2005) contains 5801 sentence pairs with a binary human judgment of whether or not each pair is a paraphrase. Three raters were used to judge the pairs according to the given guidelines. There is 82.20% inter-rater agreement between rater 1 and 2 and 84.70% agreement between rater 1 and 3. There are 4076 and 1725 pairs in the training and test sets respectively. Table 4.5 demonstrates state of the art results on the Microsoft paraphrase corpus for comparison.

Algorithm	Reference	Description	Supervision	Accuracy	F
Vector Based Similarity (Baseline)	Mihalcea et al. (2006)	cosine similarity with tf-idf weighting	unsupervised	65.4%	75.3%
ESA	Hassan (2011)	explicit semantic space	unsupervised	67.0%	79.3%
KM	Kozareva and Montoyo (2006)	combination of lexical and semantic features	supervised	76.6%	79.6%
LSA	Hassan (2011)	latent semantic space	unsupervised	68.8%	79.9%
RMLMG	Rus et al. (2008)	graph subsumption	unsupervised	70.6%	80.5%
MCS	Mihalcea et al. (2006)	combination of several word similarity measures	unsupervised	70.3%	81.3%
STS	Islam and Inkpen (2007)	combination of semantic and string similarity	unsupervised	72.6%	81.3%
SSA	Hassan (2011)	salient semantic space	unsupervised	72.5%	81.4%
QKC	Qiu et al. (2006)	sentence dissimilarity classification	supervised	72.0%	81.6%
ParaDetect	Zia and Wasif (2012)	PI using semantic heuristic features	supervised	74.7%	81.8%
Vector-based similarity	Milajevs et al. (2014)	Additive composition of vectors and cosine distance	unsupervised	73.0%	82.0%
SDS	Blacoe and Lapata (2012)	simple distributional semantic space	supervised	73.0%	82.3%
matrixJcn	Fernando and Stevenson (2008)	JCN WordNet similarity with matrix	unsupervised	74.1%	82.4%
FHS	Finch et al. (2005)	combination of MT evaluation measures as features	supervised	75.0%	82.7%
PE	Das and Smith (2009)	product of experts	supervised	76.1%	82.7%
WDDP	Wan et al. (2006)	dependency-based features	supervised	75.6%	83.0%
SHPNM	Socher et al. (2011)	recursive autoencoder with dynamic pooling	supervised	76.8%	83.6%
MTMETRICS	Madnani et al.	combination of eight	supervised	77.4%	84.1%

	(2012)	machine translation metrics			
L.D.C Model	Wang et al. (2016)	Sentence Similarity Learning by Lexical Decomposition and Composition	supervised	78.4%	84.7%
Multi-Perspective CNN	He et al. (2015)	Multi-perspective Convolutional NNs and structured similarity layer	supervised	78.6%	84.7%
REL-TK	Filice et al. (2015)	Combination of Convolution Kernels and similarity scores	supervised	79.1%	85.2%
SAMS-RecNN	Cheng and Kartsaklis (2015)	Recursive NNs using syntax-aware multi-sense word embeddings	supervised	78.6%	85.3%
TF-KLD	Ji and Eisenstein (2013)	Matrix factorization with supervised reweighting	supervised	80.4%	85.9%

Table 4.4 State of the art results for MSR

4.2.4 Word Embeddings Used in Composition Models

As word vectors we used two different sets: first, off-the-shelf word embeddings trained on huge corpora. In this case, BOW2, BOW5, and DEPS are taken from Levy & Goldberg’s (2014) paper where for DEPS, after excluding the words occurring less than 100 times, there are 175,000 words with over 900,000 distinct syntactic contexts. The second set is the embeddings we built using the Wikicorpus with 600 million words. Among all different types of embeddings we trained on the corpus, we used BOW2, BOW5, DEPS, FastText-W2. As stated in the previous section, the first two embeddings (BOW2, BOW5) were built using the BOW model (Mikolov et al., 2013), the DEPS is the syntax-based embedding suggested by Levy & Goldberg (2014), and the last two models are based on the model proposed by Bojanowski et al. (2016) with the window size of 2 and 5 respectively. All models were built using SkipGram algorithm (Mikolov et al., 2013).

4.2.5 Procedure

We began with the M&L dataset where each pair contains two phrases which in turn are composed of two words. The vector for each phrase was created using the additive model. In this model, we add word vectors to build phrase vectors as discussed in the beginning of this chapter. The similarity between the vectors of the phrases was calculated using the cosine similarity measure. Then the result was multiplied by 7 to comply with the scoring scale of the dataset. After computing the cosine similarity for all phrase pairs in the dataset, we calculated the Spearman correlation between the similarity scores produced by our system and the Gold Standard. The same steps were taken for multiplicative and concatenation models.

MSR was the other dataset we used to evaluate our compositional models with. As mentioned in the previous paragraphs, the dataset contains pairs of sentences (not two-word phrases) as opposed to M&L. Since the sentences are of variable length, we used averaging instead of summation to build each sentence's vector. To build such a vector we added the sentence's constituent words' vectors and then divided it by the number of words in the sentence. For instance, imagine the vectors for *She*, *is* and *smart* are as follows: *She* = [1 1 1 0]; *is* = [0 1 1 1]; *smart* = [1 0 1 1]; the vector for the sentence *She is smart* = [0.66 0.66 1 0.66]; the reason to use averaging instead of summation was that we did not want the model to be biased toward sentences of longer length.

We did not use the multiplication model on this dataset, because after running the experiments on the M&L dataset, we observed that this model did not yield good results. We did not use concatenation either due to the fact that sentences in each pair were of different length, therefore concatenation would have resulted in vectors of different length which were not comparable. Moreover, the concatenation model had not produced good results on M&L dataset.

The MSR dataset is binary, in order to set a cut-off point between a paraphrase and non-paraphrase we defined a number of different thresholds for each model and tested them on the training set, which we used as a development set; in each case, the threshold which yielded the highest F-score was selected for the test set. In particular, we specified the thresholds to be 0.69, 0.79, and 0.89; it means if the threshold was 0.69 and the similarity score between two sentences in a pair was bigger than 0.69 (it could be a number between 0 and 1), the pair was considered as a paraphrase.

As our evaluation metrics, we used precision, recall and F-score like the other researchers (Milajevs et al., 2014) which are reported in the next section. These metrics are defined as follows:

Precision=$\frac{TP}{TP+FP}$

Recall=$\frac{TP}{TP+FN}$

F_score=2× $\frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$

, where TP (true positive) equals the number of paraphrases in the dataset which were detected as a "paraphrase" by the system. TN (true negative) means the number of non-paraphrases which were detected as a "non-paraphrase" by the system. FP (false positive) represents the number of non-paraphrases recognized as a "paraphrase" and FN (false negative) are those pairs which were a paraphrase but was identified as a "non-paraphrase".

5 Results and Discussion

In previous chapters we argued that the type of context and the size of corpus affect the quality of the resulting word embeddings. To verify this, we built three different semantic spaces using three different contexts (bag-of-words, syntactic, and character n-gram) and two corpora of different sizes (British National Corpus and Wikipedia). We evaluated the resulting embeddings in various tasks qualitatively and quantitatively. Then we defined three composition models (summation/averaging, concatenation and multiplication) to build sentence representations using the previous word embeddings. These models are evaluated in two different tasks (phrase and sentence similarity), this chapter is divided into two parts. In the first part, we report the results for the word- embeddings and discuss them; in the second part, we deal with the results and discussion of the composition models.

5.1 Results and Discussion: Word Embeddings

Following Levy & Goldberg's (2014) tradition, we evaluated the quality of the word embeddings both qualitatively and quantitatively. In the following parts, we discuss these methods of evaluation in more details.

5.1.1 Qualitative Evaluation

In the qualitative evaluation, we calculated the most similar words to a set of target words. The cosine similarity was used to measure the similarity between the vectors and some of the target words were selected from the words chosen by Levy & Goldberg (2014). We intentionally chose some similar target words to see if the results we got were different from their results.

Table 5.1 shows five most similar words to the selected target words. The first word is, *Batman*, which shows similar sets of results across different embeddings. This is what we observed for many other target words. However, there are some other words which show obvious differences between embedding models.

In case of *Hogwarts*, which is the school of magic in the Harry Potter story, it can be easily seen that BOW contexts represent more of a similarity in topic compared to syntax-based contexts which reflect the semantic type of the target word. In other words, the first top five most similar words to *Hogwarts* in BOW contexts are words related to the topic words that all exist in the Harry Potter story. On the other hand, the first five most similar words induced by syntactic context are names of different schools having the same functionality as the word's.

The same behavior can be observed for the word *Turing*, the English scientist, and many other similar nouns. BOW models present the words that are associated with *Turing* like non-deterministic, computable, etc., while the syntactic context produces the names of other scientists. The kind of similarity represented by BOW context is described as topical similarity whereas the one by syntactic context is known as functional similarity (Turney, 2012).

Another interesting observation is for the word *Florida*. The difference between the results of different embedding models can be interpreted in terms of an ontological difference; BOW contexts induce meronyms (by showing counties or cities within *Florida*), while syntactic model provides co-hyponyms (states in US other than *Florida*). The same behavior was observed for many other geographical locations.

The last two examples illustrate that similarities resulting from syntax-based context share syntactic features. For instance, for the word *driving*, BOW models yield other forms of the verb (simple past, past participle), whereas in DEPS we see only verbs of the same form (gerund) in which the first four are related to the concept of transportation. The same applies to the word *not*, where BOW contexts induce words which imply the concept of negativity, while DEPS not only does produce the words which convey the concept of negativity, but all these words are of the same part of speech (adverb).

Finally, after inspecting many more examples, which we do not present here due to space constraints, we concluded that syntactic contexts (DEPS) induce embeddings which reflect more functional similarity between words and BOW context yield embeddings which represent more topical similarity between the words.

All in all, the DEPS model represents more functional and the BOW model more topical similarity; furthermore, as the size of the window increases (from 2 to 10) the similarity becomes more and more topical. For example, for the word *driving*, we even observe words like *car* and *driver* when the size of the window is 10 (BOW10).

It is worth mentioning that what we are reporting here is what we achieved using Wikicorpus with 600 million words because we wanted to get the results for the same words as Levy and Goldberg's and some of the words used by them were not in the BNC. However, the same quality can be seen in embeddings built with the BNC as long as the word exists in the corpus. Here, we do not report the performance of the embeddings resulting from the Brown corpus due to their low quality. We do not report the results of

FastText embeddings either, because no noticeable differences were seen between them and BOW embeddings (at least for these examples).

Target Word	BOW 2	BOW 5	BOW10	DEPS
Batman	Superman SpiderMan Batgirl Nightwing Catwoman	Superman Nightwing Batgirl Joker Catwoman	Batmans Batgirl Joker Nightwing Catwoman	Superman Catwoman Avengers Wolverine Godzilla
Hogwarts	Dumbledore Hagrid Slytherin Gryffindor Snape	Dumbledore Hagrid Malfoy Slytherin Slughorn	Dumbledore Malfoy Hagrid Weasley HalfBlood	Haileybury Bridlington Aligarh Earlham Godalming
Turing	Turings Nondeterministic Polynomialtime Gödel probabilistic	Turings Computable Nondeterministic Gödel computability	Turings computable computability nondeterministic computation	Ramanujan Gödel Feynman Faraday Dirac
Florida	Alabama Fla Tallahassee Jacksonville Sarasota	Tallahassee Jacksonville Sarasota Fla Gainesville	Tallahassee Jacksonville Tampa Floridas Sarasota	Louisiana Minnesota Oregon Michigan Colorado
driving	Drove Driven Speeding Idling drive	Drove Driven Driver Drive speeding	Drove Driven car Driving driver	Cruising Flying Walking Traveling kicking
not	NOT Indeed Isnt Wasnt never	Never Neither Indeed Nor wasnt	Neither Never Nor indeed necessarily	Never Not NOT Nt never

Table 5.1. Target words and their 5 most similar words, as captured by different embeddings.

Here, we also report the performance of the embeddings resulting from the Brown corpus, although the results are low quality still they are informative and should be provided. Table 5.2 demonstrates examples of the brown corpus results.

Target Word	BOW 2	DEPS
written	apparently served known lived established	born held drawn shown taken
not	Neither Also Never Actually indeed	never no Not Slowly together

Table 5.2. Examples of the brown corpus results

5.1.2 Quantitative Evaluation

We utilized some commonly used benchmarks to evaluate the word embeddings to see how well they resemble human judgments of semantic similarity of words. One of the most popular datasets is WordSim353 which was introduced by Finkelstein et al. in 2001. The dataset includes 353 word pairs manually annotated with a degree of similarity. For instance, *tiger-cat* is annotated with 7.35, indicating a high degree of similarity.

Agirre et al. (2009) argued that the dataset does not make a distinction between similarity and relatedness. In other words, the words in the pair *computer.keyboard* are more related than similar. Hence the dataset could not distinguish between the models which induced similarity compare to the models which reflected relatedness. Following the argument, they split the dataset into two subsets of word similarity and word relatedness where the former contains pairs which are considered similar (synonyms, antonyms, identical, or hyponym-hyperonym, etc), the latter includes related pairs (meronym- holonym, etc). We used these two subsets to evaluate the model's performance in judging the similarity and relatedness between pairs.

The other gold standard that we employed was SimLex-999. The dataset was created by Hill et al. (2014) and is larger than the previous one with 999 annotated pairs, with the aim of focusing on similarity (as opposed to the relatedness).

The resulting embeddings were evaluated by calculating the cosine similarity between the vectors of the words in each pair. Since the cosine similarity function produces a number between 0 and 1, the results were multiplied by 10 so that they could be compared with the gold standard scores. Finally, the Spearman's correlation between the rankings of word pairs induced from Gold Standard and the rankings that results from the embeddings was computed and reported. Table 5.3, illustrates a summary of the obtained results across different models.

Model	WordSim353-Similarity	WordSim353-Relateness	Simlex999
BOW2-BNC	69%	50%	38%
FastText BOW2-BNC	69%	42%	32%
DEPS-BNC	54%	18%	27%

Table 5.3. Spearman's correlation between the scores generated by different systems and the Gold Standard scores. All the models were trained with BNC which contains 100 million words.

We kept the size of the corpus fixed (only BNC) and changed the type of the context to see which one produced better representation for words for the task of similarity and relatedness. As you can see, the BOW model outperforms all other models, except for the WordSim353-Similarity task where it is on par with FastText. The DEPS (syntax-based context) performs worst in relatedness task which is in line with what has been already reported in the literature (Levy & Goldberg, 2014; Melamud et al., 2016). However, the model did not produce quality embeddings for the task of similarity either. This is not expected. From the qualitative evaluation in our study and the literature as well (Levy & Goldberg, 2014; Melamud et al., 2016), we expect the DEPS model to reflect (functional) similarity between the words yielding higher scores on WordSim353-Similarity and SimLext-999 datasets. We argue that this discrepancy might be related to the size of the corpus (the corpus used by Melamud et al. (2016) contained 10 billion words). So let us see what the results are for DEPS embeddings built using a bigger corpus (Wikipedia with 600 million words).

Model	WordSim353-Similarity	WordSim353-Relateness	Simlex999
DEPS-BNC	54%	18%	27%
DEPS-Wikicorpus	59%	18%	31%

Table 5.4. Spearman's correlation between the scores generated by DEPS model and the Gold Standard's scores.

A glance at Table 5.4 shows the improvement in the model's performance by 4 to 5 percent on similarity task; however, on the relatedness task, the model's performance is similar no matter the size of the corpus. This confirms that syntax-based (DEPS) models do not induce relatedness between words regardless of the size of the corpus they are trained with. Moreover, despite the enhancement in the model's performance in the similarity task, the model still lags behind the BOW model by a noticeable margin. Table 5.5 confirms this by illustrating the results achieved by these two models on similarity and relatedness tasks.

Model	WordSim353-Similarity	WordSim353-Relateness	Simlex999
BOW2-Wikicorpus	79%	60%	40%
DEPS-Wikicorpus	59%	18%	31%

Table 5.5. Spearman's correlation between the scores generated by different models and the Gold Standard scores. All the models were trained with Wikicorpus.

It seems that the DEPS (syntax-based) method requires much bigger corpora to induce quality embeddings. Levy & Goldberg (2014) had 175,000 words and 900,000 contexts in their vocabulary, whereas we had 92,970 and 186,953 in our word and context vocabulary respectively (in the Wikicorpus). For example, imagine the following scenario where the corpus contains only two sentences (articles are removed).

Scientist discovered star.

Star discovered by scientist.

In the BOW model with the window size of 2, *star* is considered as the context of the word *discovered* two times, while in DEPS method *discovered* is paired with *star* in two different ways. For the first sentence we have (discovered, star/dobj) and for the second sentence, we have (discovered, star/nsubjpass). So in order to have the same number of target-context co-occurrences for the syntax-based (DEPS) model we need a corpus twice as big.

Another point is that other researchers (Levy & Goldberg, 2014; Melamud et al., 2016) lowercased the corpus before building the DEPS word embeddings. This also affects the number of target-context co-occurrences.

The final point worth mentioning is that for the BOW model even though we used a smaller corpus (Wikicorpus with 600 million words), we achieved a correlation score close to what other researchers obtained using much bigger corpora (Melamud et al.,2016, Corpus of 10 Billion words). This shows that for the BOW model the size of the corpus does not affect the quality of produced embeddings after a certain level and the only reason to use huge corpora could be the issue of word coverage.

5.2 Results and Discussion: Sentence Vectors

The first series of the results are the ones obtained on the M&L dataset. Table5.6 shows the Spearman correlation between the scores produced by summative, multiplicative and concatenation models using different word embeddings as the input.

Word Model	Composition Model	Correlation
BOW2	Summation	47%
BOW5	Summation	**48%**
DEPS	Summation	45%
BOW2	Multiplication	34%
BOW5	Multiplication	35%
DEPS	Multiplication	35%
BOW2	Concatenation	45%
BOW5	Concatenation	46%
DEPS	Concatenation	40%

Table 5.6. Spearman Correlation between the scores produced by different composition models on M&L dataset. The word models are off-the-shelf embeddings trained on huge corpora.

The word models reported in Table 5.6 are those trained on the huge corpora which are available on their creators' websites. The reason to use these embeddings is that we wanted to see how the results produced by them might be different from ones produced by embeddings we built on smaller corpora.

As you can see the winner is the composition model which uses BOW5 as its word embedding model and summation as its composition function. Mitchell and Lapata (2010) used traditional models to build word vectors (word co-occurrence and topic-based). For them, the multiplication worked best when they used co-occurrence based model for the words and summation was preferable when they used word vectors built using the topic-based model. They reported the correlation coefficients for different sets of phrases (Adjective-Noun, Noun-Noun, and Verb-object) separately with 49% being the highest using Multiplication as composition model and co-occurrence based model for word vectors on Noun-Noun phrase pairs.

In the next step, we did the same experiment but this time with the word embeddings that we had created on Wikicorpus which is considered a small corpus compared to the corpora the previous word embeddings have been trained on. Here, we want to investigate the effect of the size of training corpora on the quality of the resulting word embeddings in an extrinsic task. Table 5.7 shows the Spearman correlation coefficients of composition model' predictions using word embeddings trained on a smaller corpus. At this step, the only composition model we used was summation due to its superior performance in the previous step.

Word Model	Composition Model	Correlation
BOW2-Wikicorpus	Summation	48%
BOW5- Wikicorpus	Summation	48%
DEPS-Wikicorpus	Summation	32%
FastText-W2	Summation	48%
FastText-W5	Summation	48%

Table 5.7. Spearman correlation between the scores produced by the summative model on the M&L dataset. The word models are our embeddings (BOW2,BOW5,DEPS,FastText-W2, and FastText-W5) trained on a smaller corpus.

As shown in Table 5.7, the embeddings show equal or in some cases (BOW2) even better performance (1% improvement) despite the smaller size of the corpus they were trained on. However, in the case of syntax-based context (DEPS), we observe that the performance drops by a noticeable margin (13%) which is what we expected given the results achieved before at the word level. This confirms the hypothesis that this kind of context requires big corpora to work well.

We followed our experiments by testing the quality of embeddings in the task of paraphrase detection and for that purpose, we used the MSR dataset. Table 5.8 demonstrates the results across different semantic spaces using off-the-shelf word embeddings which were built on huge corpora. Table 5.9 shows the same results but for the embeddings acquired with the Wikicorpus.

Word_Model	Composition	Threshold	Precision	Recall	F_score
BOW5	Averaging	>0.79	**70%**	95%	80%
BOW2	Averaging	>0.79	68%	97%	80%
DEPS	Averaging	>0.79	67%	99%	80%
wiki-news-1MFastText.vec	Averaging	>0.89	67%	**100%**	80%

Table 5.8. The precision, recall, and F-score for different semantic spaces, Wikipedia and FastText using off-the-shelf word embeddings.

As you can see there is not any difference between the results achieved by different semantic spaces and all show the same F-scores. All models have a high recall (95% to 100%) and acceptable precision score (67% to 70%). It means that the models can return most of the paraphrases (95% to 100%) and 67% to 70% of what they return are actually paraphrased.

Word_Model	Composition	Threshold	Precision	Recall	F_score
BOW2	Averaging	>0.79	68%	99%	80%
BOW5	Averaging	>0.79	67%	**100%**	80%
DEPS	Averaging	>0.89	68%	98%	80%
FastText2	Averaging	>0.79	68%	99%	**81%**
FastText5	Averaging	>0.89	**73%**	91%	**81%**

Table 5.9. The precision, recall, and F-score for different semantic spaces using the word embeddings built with Wikicorpus (the small corpus).

Interestingly, decreasing the size of the corpus the embeddings were built with has led to a better performance and FastText embeddings with window sizes of 2, and 5 both show 81% F-score. The FastText embeddings in Table 5.8 were built by combining Wikipedia and Google news corpora, whereas the embeddings in Table 5.9 are the result of a much smaller corpus (Wikicorpus with 600 million words). One reason could be the quality of the text in the corpus (Wikicorpus is very clean) and the topic it covers. Moreover, one of the features of the FastText model is to produce high-quality word embeddings with small corpora which is what we observed in this experiment. However, it is worth mentioning that building the word embeddings using BOW was much faster than building them through FastText. It took 8 hours for us to build BOW5 embeddings with Wikicorpus corpus while the same process took more than 3 days for the FastTextW5 model.

6 Conclusion and Future work

We built different word and phrase/sentence embeddings using different contexts and corpora of various sizes. We observed that syntax-based (DEPS) embeddings induce functional similarity when we evaluated it qualitatively; however, it needs to be trained on bigger corpora to demonstrate the same quality in quantitative evaluation. Character N-gram (FastText) proved to be less dependent on the size of the corpus and produces high-quality embeddings with a corpus of a small size. However, building word embeddings using FastText is very time consuming especially when the size of the window increases. BOW is the fastest and least time-consuming model to build.

Another point is that representing a sentence by averaging the vectors of its constituent words is a robust composition model. This type of sentence representation works well across different tasks (Wieting et al., 2015) including paraphrase detection. Moreover, FastText embeddings are the most suited embeddings to represent the sentence's constituent words in the task of paraphrase detection, especially when the size of the corpus is small.

In the future, we would like to continue working on semantic-based word embeddings as mentioned in the methodology chapter. We would like to know what kind of semantic relation can be captured using this kind of context. We would also like to explore the potential of multilingual word embeddings in distinguishing between synonyms, and other types of relations such as antonyms and (co)hyponyms (van der Plas and Tiedemann, 2006). This is an important quality which cannot be captured by the above-mentioned embedding models.

As for the composition models, we would like to investigate the possibility of a general composition model that works across domains and will not be task specific, unlike neural-based composition models. This model will aim to take into account the order of words in the sentence. It is an important quality, especially when we consider sentences like "*John laughed at Mary*." and "*Mary laughed at John*." Our simple composition models in this thesis fail to understand that these two sentences are opposite and consider them similar.

References

Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Paşca, M., & Soroa, A. (2009). A study on similarity and relatedness using distributional and wordnet-based approaches. In Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (pp. 19-27). Association for Computational Linguistics.

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K, Collins, M. (2016). Globally Normalized Transition-Based Neural Networks. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). doi:10.18653/v1/p16-1231

Baroni, M. (2013). Composition in distributional semantics. Language and Linguistics Compass, 7(10), 511-522.

Baroni, M., & Zamparelli, R. (2010). Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (pp. 1183-1193). Association for Computational Linguistics.

Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Vol. 1, pp. 238-247).

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning, research, 3,1137-1155.

Blacoe, W., & Lapata, M. (2012). A comparison of vector-based representations for semantic composition. In Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning (pp. 546-556). Association for Computational Linguistics.

Blei, D.M., Ng, A.Y. and Jordan, M.I. (2003) Latent Dirichlet Allocation. The Journal of Machine Learning Research, 3, 993-1022.

Bojanowski, P., Grave, E., Julian, A., & Mikolov, T. (2016). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistic (TACL), 5:135–146..

Bullinaria, J. A., & Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. Behavior research methods, 39(3), 510-526.

Clark, S. (2015). Vector space models of lexical meaning. The Handbook of Contemporary semantic theory, 493-522.

Coecke, B., Sadrzadeh, M., & Clark, S. (2010). Mathematical foundations for a compositional distributional model of meaning/. Lambek Festschrift Linguistic Analysis, 36. 345–84.

Collobert, R. (2011).Deep Learning for Efficient Discriminative Parsing, in International Conference on Artificial Intelligence and Statistics (AISTATS).

Cornea, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017, pages 670– 680.

Dolan, B., Quirk, C., & Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In Proceedings of the 20th international conference on Computational Linguistics(p. 350). Association for Computational Linguistics.

Dozat, T., & Manning, C. D. (2017). Deep biaffine attention for neural dependency parsing. ICLR.

Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., & Smith, N. A. (2015). Retrofitting word vectors to semantic lexicons. 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, pages 1606–1615. .

Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. CoRR, abs/1402.3722, 2014..

Grefenstette, E., & Sadrzadeh, M. (2011). Experimenting with transitive verbs in a discocat. In Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics (pp. 62-66). Association for Computational Linguistics.

Harris, Z. S. (1954). Distributional structure. Word, 10(2-3), 146-162.

Hill, F., Reichart, R., & Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. Computational Linguistics, 41(4), 665-695.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. Psychological review, 104(2), 211.

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In International Conference on Machine Learning (pp. 1188-1196).

Leech, R. Garside, and M. Bryant. (1994). CLAWS4: The tagging of the British National Corpus. In Proceedings of the 15th International Conference on Computational Linguistics (COLING), pages 622-628, Kyoto, Japan.

Lemaire, B., & Denhiere, G. (2006). Effects of high-order co-occurrences on word semantic similarity. Current psychology letters. Behavior, brain & cognition, (18, Vol. 1, 2006).

Levy, O., & Goldberg, Y. (2014). Dependency-based word embedding. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (Vol. 2, pp. 302-308).

Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving distributional similarity with lessons learned from word embedding. Transactions of the Association for Computational Linguistics, 3, 211-225.

Marcus, M. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. doi:10.21236/ada273556

Melamud, O., McClosky, D., Patwardhan, S., & Bansal, M. (2016). The role of context types and dimensionality in learning word embeddings. . In NAACL-HLT, pages 1030–1040.Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

Mitchell, J., & Lapata, M. (2008). Vector-based models of semantic composition. proceedings of ACL-08: HLT, 236-244.

Mitchell, J., & Lapata, M. (2010). Composition in distributional models of semantics. Cognitive science, 34(8), 1388-1429.

Padó, S., & Lapata, M. (2007). Dependency-based construction of semantic space models. Computational Linguistics, 33(2), 161-199.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

Quirk, C., Brockett, C., & Dolan, B. (2004). Monolingual machine translation for paraphrase generation, In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, Barcelona Spain.

Reese, S., Boleda Torrent, G., Cuadros Oller, M., Padró, L., & Rigau Claramunt, G. (2010). Word-sense disambiguated multilingual Wikipedia corpus. In 7th International Conference on Language Resources and Evaluation.

Rehurek, R., & Sojka, P. (2010). Software framework for topic modeling with large corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.

Sahlgren, M. (2006). The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces (Doctoral dissertation).

Socher, R., Huang, E. H., Benin, J., Manning, C. D., & Ng, A. Y. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In Advances in neural information processing systems (pp. 801-809).

Socher, R., Huval, B., Manning, C. D., & Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning (pp. 1201-1211). Association for Computational Linguistics.

Turney, P. D. (2012). Domain and function: A dualspace model of semantic relations and compositions. Journal of artificial intelligence research, 44,533– 585.

Turney, P. D., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. Journal of artificial intelligence research, 37, 141-188.

Van der Plas, L., & Tiedemann, J. (2006). Finding synonyms using automatic word alignment and measures of distributional similarity. In Proceedings of the COLING/ACL on Main conference poster sessions (pp. 866-873). Association for Computational Linguistics.

Wieting, J., Bansal, M., Gimpel, K., & Livescu, K. (2015). Towards universal paraphrastic sentence embedding. In International Conference on Learning Representations (ICLR), 2016a