applied sciences

MDPI

Article

A Polarity Capturing Sphere for Word to Vector Representation

Sandra Rizkallah *, Amir F. Atiya and Samir Shaheen

Faculty of Engineering, Department of Computer Engineering, Cairo University, Giza Governorate 12613, Egypt; amir@alumni.caltech.edu (A.F.A.); sshaheen@eng.cu.edu.eg (S.S.)

* Correspondence: sandrawahid@hotmail.com

Received: 20 May 2020; Accepted: 23 June 2020; Published: 26 June 2020

Abstract: Embedding words from a dictionary as vectors in a space has become an active research field, due to its many uses in several natural language processing applications. Distances between the vectors should reflect the relatedness between the corresponding words. The problem with existing word embedding methods is that they often fail to distinguish between synonymous, antonymous, and unrelated word pairs. Meanwhile, polarity detection is crucial for applications such as sentiment analysis. In this work we propose an embedding approach that is designed to capture the polarity issue. The approach is based on embedding the word vectors into a sphere, whereby the dot product between any vectors represents the similarity. Vectors corresponding to synonymous words would be close to each other on the sphere, while a word and its antonym would lie at opposite poles of the sphere. The approach used to design the vectors is a simple relaxation algorithm. The proposed word embedding is successful in distinguishing between synonyms, antonyms, and unrelated word pairs. It achieves results that are better than those of some of the state-of-the-art techniques and competes well with the others.

Keywords: word to vector; word embeddings; antonymy detection; polarity

1. Introduction

Word vector embeddings seek to model each word as a multi-dimensional vector. There are many distinct benefits to modeling words as vectors. Implementing natural language processing (NLP) algorithms using machine learning models necessitates converting a textual word or sentence to a numeric format. Moreover, this conversion has to be meaningful. For example, words with similar meanings should possess vectors that are in close proximity in the embedded space. This is because such words should produce similar outputs, if applied to a machine learning model. Additionally, a word embedding of a complete dictionary will provide a complete representation of the corpus. In this, each word is assigned to its unique position in the vector space, which reflects its aggregate relations with all other words in one cohesive construct. Word vector spaces have been very useful in many applications; for example, machine translation [1], sentiment analysis [2–5], question answering [6,7], information retrieval [8,9], spelling correction [10], crowdsourcing [11], named entity recognition (NER) [12], text summarization [13–15], and others.

The problem of existing word vector embedding methods is that the polarity of words is not adequately considered. Two antonyms are considered polar opposites, and have to be modeled as such. However, most current methods do not deal with this issue. For example, they do not differentiate in the similarity score between two unrelated words and two opposite words.

In this work we propose a new embedding that takes into account the polarity issue. The new approach is based on embedding the words into a sphere, whereby the dot product of the corresponding vectors represents the similarity in meaning between any two words. The polar nature of the sphere is

the main motivation behind such embedding since this way antonymous relations can be captured such that a word and its antonym are placed at opposite poles of the sphere. This polarity capturing feature is essential for some applications such as sentiment analysis. Recently, word embedding methods have started to pervade the sentiment analysis field, at the expense of traditional machine learning algorithms, which rely on sentiment-polarity data that are annotated manually, and expensive feature engineering. Using word embeddings, a sentence can be mapped to a number of features based on the embeddings of its words. However, these embeddings should reflect the polarity of words in order to be able to perform the sentiment analysis task. As mentioned before, this polarity issue is completely addressed in our word embedding approach.

Figure 1 illustrates the concept of the proposed approach. As shown in the figure, the two vectors corresponding to the words "happy" and "joyful" have close similarity because they are synonyms. On the other hand, the vectors corresponding to "happy" and "sad" are on opposite poles and have a similarity score of -1, because they are antonyms. The algorithm can infer a new relation that "joyful" and "sad" are antonyms too, as their corresponding vectors are also placed close to opposite poles of the sphere.

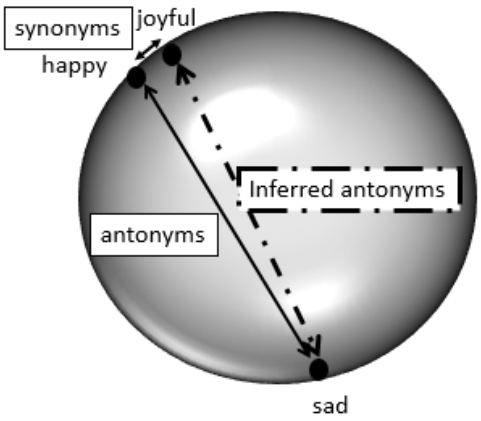

Figure 1. Semi-supervised illustration.

Two words that are similar in meaning are assigned vectors whose similarity measure is 1. Two unrelated words will be assigned a similarity close to zero, and two antonyms are assigned similarity close to -1. In other words, we specifically consider the polarity issue and make a distinction between "opposite" and "unrelated".

The fact that the similarity between opposing words or antonyms is close to -1 is consistent with the logically appealing fact that negation amounts to multiplying by -1. Double negation amounts to multiplying -1 twice, giving a similarity close to one (the antonym of an antonym is a synonym). In our geometry, negation can also be considered as reflection around the origin. Thus, the designed model makes logical sense. Another question to investigate is whether the unrelated words should have similarity close to zero. The answer is yes. It is logical that unrelated words, e.g., "train" and "star", would have their vectors far away from each other, because of our principle that vector similarities or closeness should reflect relatedness. This is consistent with the theory that at a high dimension, the dot product of randomly occurring unit vectors on the sphere is close to zero [16]. The algorithm has a number of beneficial features:

- The proposed algorithm is a very simple relaxation algorithm, and it converges very well and fast. It is simpler than the typically used word to vector design methods, such as neural networks or deep networks.
- We believe that embedding the vectors into a sphere provides a natural representation, unlike distributional models for learning word vector representations and other co-occurrence-based models. Some of these models often fail to distinguish between synonyms and antonyms,

since antonymous words occur in similar context the same way synonymous words do. Other models fail to distinguish between antonymous word pairs and unrelated word pairs.

- The learned word vectors can always be continuously augmented. The database can flexibly be extended to include more words by simply feeding the algorithm with more word pairs with the desired target similarity scores. The algorithm guarantees that these new vectors will be placed correctly with regard to the existing vectors.
- Composite words can also be handled by our algorithm. This is because any word pair with the desired similarity score can be fed into the algorithm . For example, "old-fashioned" is an expression that combines two words: "old" and "fashion"; this combined expression can be given to our algorithm as a synonym to a word that conveys its meaning; e.g., "outdated".
- Generally it is a supervised procedure. The designer typically gives synonyms/antonyms from thesauri or other sources. However, it could also be designed in a semi-supervised setting. This means that in addition to the user-defined similarity scores between the collected words, there could be many more unlabeled pairs that have to be learned from the text pieces available (for example, through co-occurrence arguments). The labeled pairs will provide some anchor around which the unlabeled pairs will organize, and are therefore essential for guiding the correct vector placements. In other words, new relations between word pairs will be inferred from existing ones.

2. Literature Review

One of the earliest word vector embeddings is the work by Mikolov et al. [17], who published the Google word2vec representations of words. Their method relies on the continuous skip-gram model for learning the vectors introduced in [18]. Finding vector representations that predict the surrounding words is the training goal of the skip-gram model. For computational efficiency, the authors have replaced the full softmax with three alternative choices, and evaluated the results for each: hierarchical softmax, noise contrastive estimation, and negative sampling. Moreover, subsampling of frequent words was introduced to speed up the training and leverage the accuracy of word representations.

Pennington et al. [19] have published another approach (called GloVe) of word vector modeling using a method that combines the benefits of the techniques of global matrix factorization and local context window. The authors have suggested the idea of learning the word vector with the ratios of co-occurrence probabilities rather than the probabilities themselves. The result is a new global log-bilinear regression model, where the model directly captures global corpus statistics.

Mikolov et al. [20] developed a model known as fastText that is based on continuous bag-of-words (cbow) used in word2vec [18]. The authors used a combination of known improvements to learn high-quality word vector representations. To obtain higher accuracy, the authors added position-dependent weights and subword features to the cbow model architecture.

Dev et al. [21] developed a technique that aligns word vector representations from different sources such as word2vec [17] and GloVe [19]. The authors extend absolute orientation approaches to work for arbitrary dimensions.

It can be noted that word2vec [17,18], GloVe [19], and fastText [20] are distributional models for learning word vector representations. Words are assigned similarity relations based on co-occurrence in text. The limitation of these models is a weakness in distinguishing antonyms from synonyms because antonymous words such as "good" and "bad" very often occur in similar contexts so their learned vectors will be close. This way it may be hard to figure out whether a pair of words is a pair of synonyms or antonyms. However, such information is crucial for some applications such as sentiment analysis. On the other hand, our approach is a simple relaxation algorithm that takes into account learning word vectors that are able to distinguish between word relations such as synonyms, antonyms, and unrelated words. It is based on embedding the vectors on the sphere. The sphere provides a natural setting for this task, because antonyms can be placed on opposite sides of the sphere. In contrast, most methods embed the vectors in the space R^n, which does not provide a natural way of

modeling opposites. The aforementioned models rely on corpus data of huge sizes to guarantee the effectiveness of the generated vectors. In our approach, we make use of the available experts' lexicons, dictionaries, and thesauruses, and so the similarity relations are fairly accurate, because of the expert knowledge used to construct these lexicons.

Other works include Vilnis et al. [22], who introduced density-based distributed embeddings by learning representations in the space of Gaussian distributions. Bian et al. [23] focused on incorporating morphological, syntactic, and semantic knowledge with deep learning to obtain more efficient word embeddings. Zhou et al. [24] used the category information associated with the community question answering pages as metadata. These metadata are fed to the skip-gram model to learn continuous word embeddings, followed by applying Fisher kernel to obtain fixed length vectors.

Some works introduce retrofitting to overcome the deficiencies in distributional models in representing the semantic and relational knowledge between words. Faruqui et al. [25] applied retrofitting to pre-trained word vectors from distributional models. The vectors are refined to account for the information in semantic lexicons. The method used is graph-based learning where the graph is constructed for the relations extracted from the lexicons. Jo [26] proposed extrofitting by extracting the semantic relations between words from their vectors using latent semantic analysis. The author further combined extrofitting with external lexicons for synonyms to obtain improved results.

There have been some attempts at tackling the polarity issue. Mohammad et al. [27] proposed an empirical none-word embedding approach for the detection of antonymous word pairs. The authors' approach relies on the co-occurrence and distributional hypotheses of antonyms, stating that antonymous word pairs occur in similar contexts more often than chance. Nevertheless, these hypotheses are only useful indications but not sufficient conditions to detect antonymous words.

Lobanova [28] proposed pattern-based methods to automatically identify antonyms. The author pointed out how antonyms are useful in many NLP applications, including contradiction identification, paraphrase generation, and document summarization.

Yih et al. [29] derived the word vector representations using latent semantic analysis (LSA), with assigning signs to account for antonymy detection, and devising the polarity inducing LSA (PILSA). Yet the authors pivoted on the fact that words with least cosine similarity are indeed opposites without regard to distinguishing between unrelated word pairs, which also have low cosine similarity, and antonymous word pairs. To embed out-of-vocabulary words, the authors adopted a two stage strategy: first conducting a lexical analysis, and, if no match was found, using semantic relatedness. This strategy weakens the smoothness of extending their approach.

Again, Mohammad et al. [30] devised an empirical method that marks word pairs that occur in the same thesaurus category as synonyms and others that occur in contrasting categories as opposites. They then apply postprocessing rules, since based on their method one word pair may be marked as both synonym and antonym at the same time. The approach devised is a none-word embedding one.

Chang et al. [31] introduced multi-relational latent semantic analysis (MRLSA) that extends PILSA [29], modeling multiple word relations. The authors proposed a 3-way tensor, wherein each slice captures one particular relation. However, the model performance depends on the quality of a pivot slice (e.g., the synonym slice), which MRLSA has to choose. Motivated by this approach, Zhang et al. [32] introduced a Bayesian probabilistic tensor factorization model. Their model combined both thesauri information and existing word embeddings, though their model used pre-trained word embeddings.

Santus et al. [33] devised a new average-precision-based measure to discriminate between synonyms and antonyms. The measure is built on the paradox of "simultaneous similarity and difference between the antonyms." The authors deduced that both synonyms and antonyms are similar in all dimensions of meaning except one. This different dimension can be identified and used for the discrimination task.

Ono et al. [34] trained word embeddings to detect antonyms. They introduced two models: a word embeddings on thesauri information (WE-T) model and a word embeddings on thesauri and

distributional information (WE-TD) model. For WE-T, the authors applied an AdaGrad online learning method that uses a gradient-based update with automatically-determined learning rate. For WE-TD, the authors introduce skip-gram with negative sampling (SGNS). However, their model is trained such that synonymous and antonymous pairs have high and low similarity scores respectively. This imposes a challenge in differentiating between antonymous word pairs and unrelated word pairs since both will have low similarity scores.

Nguyen et al. [35] proposed augmenting lexical contrast information to distributional word embeddings in order to enhance distinguishing between synonyms and antonyms. Moreover, the authors extended the skip-gram model to incorporate the lexical contrast information into the objective function.

Motivated by the fact that antonyms mostly lie at close distances in the vector space, Li et al. [36] proposed a neural network model that is adapted to learn word embeddings for antonyms. These embeddings are used to carry out contradiction detection.

In most of these aforementioned antonymy construction methods the problem is that they focus mostly on this task only. For example [29,32,34] only evaluated their vectors on antonymy detection without showing the performance of these vectors on synonyms or unrelated words. The goal in these works has mainly not been to obtain a global word embedding that works for synonyms, antonyms, and unrelated words.

Word embeddings obtain the whole picture, as to the semantic relations of the words in a corpus. As such, they have many applications. For example, Zou et al. [1] proposed a method to learn bilingual embeddings to perform a Chinese–English machine translation task. Moreover, word embeddings methods are applied to sentiment analysis; Maas et al. [2] proposed a model that combines supervised and unsupervised techniques to learn word vectors that capture sentiment content. The goal was to be able to use these vectors in sentiment analysis tasks. The authors used an unsupervised probabilistic model of documents followed by a supervised model that maps a word vector to a predicted sentiment label using a logistic regression predictor that relies on sentiment annotated texts. Tang et al. [3] developed a word embedding method by training three neural networks. The method relies on encoding sentiment information in the continuous representations of words. They evaluated their method on a benchmark Twitter sentiment classification dataset. Dragoni et al. [4] employed word embeddings and deep learning to bridge the gap between different domains, thereby building a multi-domain sentiment model. Deho et al. [5] used word2vec to generate word vectors that learn contextual information. To perform sentiment analysis, the generated vectors were used to train machine learning algorithms in the form of classifiers. Question answering is another application of word embeddings; for example, Liang et al. [6] tackled the rice FAQ (frequently asked question) question-answering system. The authors proposed methods based on word2vec and LSTM (long-short term memory). The core of the system is question similarity computing, which is used to match users' questions and the questions in FAQ. Liu et al. [7] designed a deep learning model based on word2vec to find the best answers to the farmers' questions. Search service also exploits word embeddings; Liu et al. [9] established a model based on word embeddings to improve the accuracy of data retrieval in the cloud. Spelling correction can also be done using word embeddings. Kim et al. [10] proposed a method of correcting misspelled words in Twitter messages by using an improved Word2Vec. The authors in [11] proposed crowdsourcing where the relevance between task and worker is obtained. The proposed model involves the computation of the similarity of word vectors and the establishment of the semantic tags similar matrix database based on the Word2vec deep learning. Habibi et al. [12] proposed a method based on deep learning and statistical word embeddings to recognize biomedical named entities (NER), such as genes, chemicals, and diseases. Word embeddings have also tackled the field of text summarization; to achieve automatic summarization Kågebäck et al. [13] proposed the use of continuous vector representations as a basis for measuring similarity. Rossiello et al. [14] proposed a centroid-based method for text summarization that exploits the compositional capabilities of word embeddings.

Word embeddings have been developed for other languages as well. Zahran et al. [37] compared diverse techniques for building Arabic word embeddings, and evaluated these techniques using intrinsic and extrinsic evaluations. Soliman et al. [38] proposed the AraVec model, a pre-trained distributed word embedding project, which makes use of six different models. The authors described the used resources for building such models and the preprocessing steps involved.

3. The Proposed Method

Let $x_i = (x_{i1}, \ldots, x_{iN})^T$ be an N-dimensional vector of unity length that represents word i; i.e., $\|x_i\| = 1$. The fact that the length of the vector is 1 means that the word is embedded on a sphere. The dot product between two vectors $x_i^T x_j$ represents the similarity between their corresponding words. For example $x_i^T x_j$ for the words *easy* and *simple* would be very high (close to 1). For the words *easy* and *manageable* it would also be high, but a shade lower. For the opposites *easy* and *difficult* it would be close to -1, and for unrelated words, such as *easy* and *cat*, it would be close to zero. It is well known that when picking up any two random vectors on a sphere of high dimension, their dot product will be close to zero. This means that the bulk of the words will have similarity close to zero with the word *easy*. Note also that because of the unit length property, the dot product $x_i^T x_j$ and the distance are related in a one to one way (because $\|x_i - x_j\|^2 = \|x_i\|^2 + \|x_j\|^2 - 2x_i^T x_j = 2\left[1 - x_i^T x_j\right]$).

We collect a number of words for which we estimate a similarity number, on a scale from -1 to 1. For example, for a pair of words with corresponding vectors x_i and x_j, let the estimated similarity be s_{ij}. We collect a very large training set, obtained from some well-known synonym and antonym lists. For any word relations that are not covered by these lists, we add a moderately sized but typically not-large training set, mainly labeled by an expert human. The expert-labeled dataset serves as the nucleus that will guide the training using the other larger collected datasets.

We develop an algorithm that estimates the vectors x_i that would yield the similarity numbers as close as possible to the given similarity numbers. We define the following error function:

$$E = \sum_i \sum_j w_{ij} \left[x_i^T x_j - s_{ij}\right]^2$$
$$\text{subject to } \|x_i\| = 1, i = 1, \ldots, K \tag{1}$$

where w_{ij} is a weighting coefficient representing the confidence in the similarity estimate s_{ij}. For example, the word pairs labeled by an expert may have higher weighting coefficient than other word pairs in the remaining larger training set. It is hard to solve this large optimization problem if one seeks to obtain all K vectors x_i at once. However, we propose a relaxation algorithm, that tackles one x_i at a time. In this algorithm we focus on some x_k and fix all other x_l's for the time being. Then we optimize E with respect to x_k. This is feasible and gives a close form solution. Then, we move on to the next vector, and fix the others, optimize with respect to that vector. We continue in this manner until we complete all vectors. Then we start another cycle, and re-optimize each x_k, one at a time. We perform a few similar cycles until the algorithm converges.

Consider that we are focusing now on vector x_k, while keeping all other vectors constant. Then we can decompose the objective function in (1) into a component containing x_k and other components that do not have x_k in them, as follows.

$$E = \sum_{i \neq k} \sum_{j \neq k} w_{ij} \left[x_i^T x_j - s_{kj}\right]^2 + 2 \sum_{j \neq k} w_{kj} \left[x_k^T x_j - s_{kj}\right]^2$$
$$= R_k + 2 \sum_{j \neq k} w_{kj} \left[x_k^T x_j - s_{kj}\right]^2 \tag{2}$$

where R_k is the term not containing x_k. We skipped the term $x_k^T x_k - s_{kk}$ because it equals zero always (the similarity between a vector and itself is 1, and the norm of any vector is enforced as 1 too). The factor 2 in the RHS is to account for the existence of x_k in first summation, and in the second summation. The optimization will now focus on optimizing the second term in the summation. Using a Lagrange multiplier to take into account the unity norm constraint, we formulate the augmented objective function:

$$V = R_k + 2\sum_{j\neq k} w_{kj}\left[x_k^T x_j - s_{kj}\right]^2 - \lambda\left(x_k^T x_k - 1\right) \tag{3}$$

where λ is the Lagrange multiplier. For simplicity, let us redefine quantities in a way to skip the factor of 2 in the equation. Simplifying, we get:

$$\begin{aligned} V &= R_k + \sum_{j\neq k} w_{kj}\left[x_k^T x_j x_j^T x_k - 2x_j^T s_{kj} x_k + s_{kj}^2\right] \\ &\quad -\lambda x_k^T x_k + \lambda \\ &= R_k + x_k^T\left(\sum_{j\neq k} w_{kj} x_j x_j^T\right) x_k - \left(2\sum_{j\neq k} w_{kj} x_j^T s_{kj}\right) x_k \\ &\quad + \sum_{j\neq k} w_{kj} s_{kj}^2 - \lambda x_k^T x_k + \lambda \\ &= R_k + x_k^T A x_k - 2b^T x_k + c \end{aligned} \tag{4}$$

where we isolated x_k from the terms that do not contain x_k, and the following matrices are defined as:

$$A = \sum_{j\neq k} w_{kj} x_j x_j^T - \lambda I \tag{5}$$

$$b = \sum_{j\neq k} w_{kj} x_j s_{kj} \tag{6}$$

$$c = \sum_{j\neq k} w_{kj} s_{kj}^2 + \lambda \tag{7}$$

Taking the derivative with respect to x_k and equating to zero, we get

$$\frac{dV}{dx_k} = 2Ax_k - 2b = 0 \tag{8}$$

$$x_k = A^{-1} b \tag{9}$$

To evaluate λ we enforce the condition $x_k^T x_k = 1$.

$$x_k^T x_k = b^T A^{-2} b = 1 \tag{10}$$

where we used the fact that A is a symmetric matrix. We get

$$b^T\left[A' - \lambda I\right]^{-2} b = 1 \tag{11}$$

where

$$A' = \sum_{j\neq k} w_{kj} x_j x_j^T \tag{12}$$

To solve Equation (11), we note that λ is a scalar, so we simply implement a one-dimensional search.

Once we obtain x_k as above, we turn our attention to the next vector, and apply similar analysis. Once we complete all vectors, we perform another cycle through all vectors, and so on. The algorithm converges, in the sense that each step leads to a reduction in the objective function (1), leading to a local minimum (akin to neural network training and other machine learning algorithms). This is proven in the theorem described below.

Theorem 1. *Let the errors before and after applying (9) and (11) be E_1 and E_2 respectively (we mean the errors given by Equation (1)).*

Then

$$E_2 \leq E_1 \tag{13}$$

The application of the steps with cycling through the vectors one by one, and repeating the cycles several times will lead to a convergence of the attained vectors to some limiting values.

The proof of this assertion, as well as other details of the optimization problem, are given in the Appendix A.

Figure 2 is a graph which empirically shows that the vectors stabilize one cycle after another. Shown in the graph is $\max(|x_k(new) - x_k(old)|)$ against cycle number. The maximum is computed over the components of the vector and over all vectors x_k. One can observe that after around 15 cycles the changes in vectors become very small, indicating their convergence to their particular positions in the space.

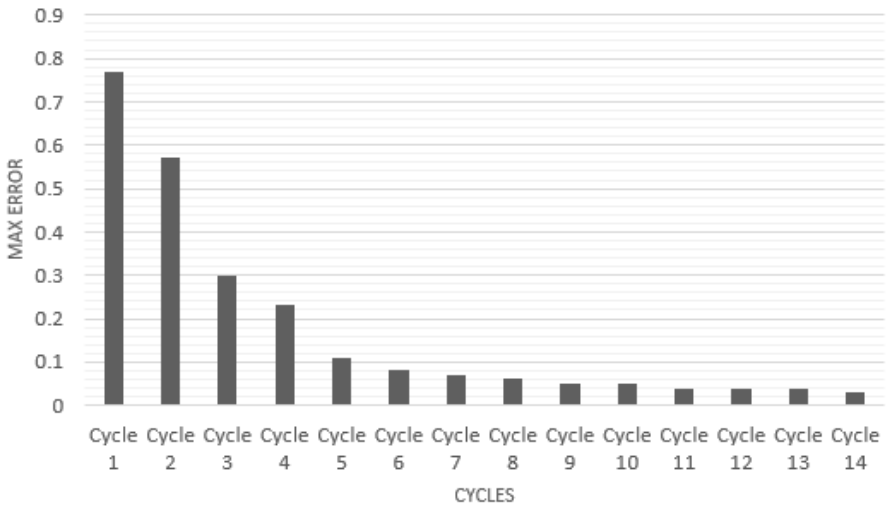

Figure 2. Convergence.

4. Vocabulary and Data Gathering

To design the proposed word embedding, we have collected data in the form of labeled pairs of words from multiple sources. Moreover, we explored the properties of the vocabulary that achieve best results using our algorithm. We model our vocabulary as a graph referred to as the graph of words, such that the words are the vertices and relations between the words are the edges. (graph modeling has been a very useful tool in natural language processing; see [39].) Weights are attached to the edges of the graph, and these weights represent the similarity scores between the two words they each connect. The existence of an edge means that a labeled word pair exists as a part of the vocabulary to train the algorithm. There is, however, a potential problem. The graph could have a number of components that are disconnected from each other; i.e., no sequence of edges can lead from one component to the other. This could lead to potentially multiple solutions for the optimization task. The algorithm would not know where to place the vectors corresponding to the disconnected

components with respect to each other. In other words, one can rotate entire connected groups without them affecting the error function (in Equation (1)), because no similarity terms exist between any of the disconnected groups' vertices. This would lead to arbitrarily estimated similarities between the vectors (words) of any two disconnected components.

Venkatesh p. 124 [40] provides an in depth investigation of when a large graph is void of disconnected components; i.e., one large connected component. He proves the following theorem:

Theorem 2. *Consider a graph with n vertices and with probability p that an edge exists between any two vertices. If $p = \frac{\log n}{n} + \frac{c}{n}$, for some constant c, then the probability that the random graph $G(n, p)$ is connected tends to $e^{-e^{-c}}$, as $n \longrightarrow \infty$.*

This theorem essentially says that if $p >> \log(n)/n$ then the graph is one giant connected component. Since the expected degree of each vertex is $k = np$, this means that the average degree of our graph should be generally higher than $\log(n)$; i.e., $k > \log(n)$. We used this fact as a guide in determining the number of synonym/antonym pairs used to create the training set, since each pair will create an edge in the out graph of words.

At the end of generating the training set, we apply the networkx python algorithm [41] to detect the number of components in a graph. There may still be several disconnected components. In such a case, we manually select pairs of words corresponding to the disconnected components. We seek to connect them by estimating the similarity using our judgment of the word meanings. This essentially draws edges between the disconnected components. Ideally, how many edges should we add between any two disconnected components? The answer is d, the dimension of our vector space. This would essentially anchor the vectors in fixed places with respect to each other. It would also prevent the possibility of rotating a component with respect to another along some remaining degrees of freedom without violating the existing distances or similarities between the vectors as given in the training set.

To collect our vocabulary we have used several sources. This is in order not to rely overly on one single source. We used the following:

- Lists for frequent English words' synonyms and antonyms extracted from the web sites: [42,43].
- Lists for certain categories that we manually constructed; e.g., family, sports, animals, countries, capitals, and others (we added 33 lists). Each list consists of many words, and we assigned a specific similarity score among the words in each list, based on our judgment.
- We collected an extensive amount of words from WordNet [44] and from other sources, such as educational websites and books [45–53]. Subsequently, we created a crawler that would visit the site of thesaurus.com (the premier site for word meanings, synonyms, antonyms, etc.) [54]. The crawler would fetch the synonyms and antonyms of the sought words from the thesaurus. The obtained words would be fed again to the site and more synonyms and antonyms were obtained, and so on. These would then be added to the training set.
- We gathered random word pairs from a site that contains random phrases [55]. We checked these pairs, and selected only the ones that were unrelated. As mentioned before, unrelated pairs should give similarity around zero, and they have the important task of connecting disconnected groups in the graph. In addition to the unrelated pairs, we collected synonyms and antonyms for these selected words using the crawler from thesaurus.com.
- We gathered other unrelated word pairs randomly by pairing words manually in the constructed vocabulary (of course after checking that they are indeed unrelated).
- We manually added some synonyms, antonyms, and unrelated word pairs using other different online dictionaries.

For the collected pairs we have assigned a similarity close to 1 for synonyms and close to -1 for antonyms. This is just the theoretical target function. After convergence it typically yields different similarities. The reason is that there is competition between words to pull the vectors of its synonymous

words towards its vector. This results in "middle ground" vector locations that satisfy reasonable contiguity towards its different synonyms. Table 1 shows the structure of the constructed vocabulary while Figure 3 is a histogram for the degree of vertices in the constructed vocabulary.

Table 1. Vocabulary structure.

Number of distinct words (graph vertices)	27770
Number of synonymous pairs	102260
Number of antonymous pairs	38783
Number of unrelated pairs	182287
Number of graph edges	323330
Ratio of number of edges to number of vertices	11.64
Average degree of the graph	23.29
Number of connected components	1

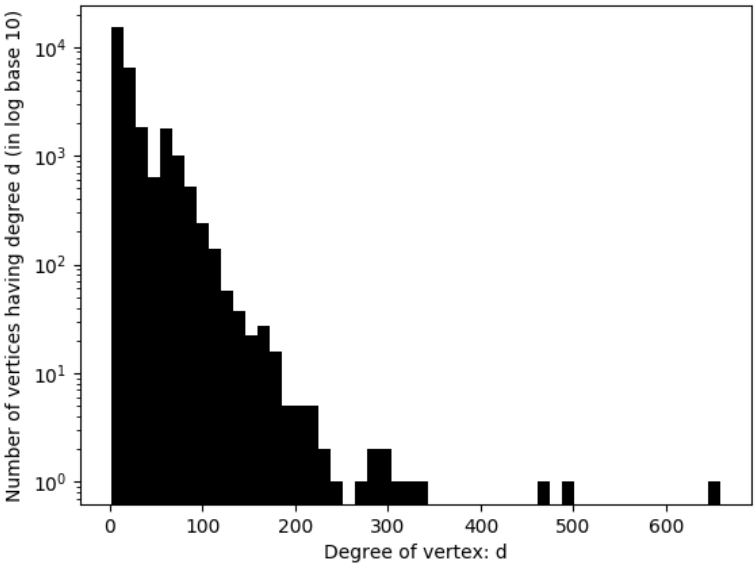

Figure 3. Histogram for the degree of vertices.

5. Results and Discussion

We have tested the proposed word embedding method. After a thorough design, and training using our method, we have performed the following four evaluation experiments in addition to outlining an opinion mining application for hotel reviews:

1. Word Similarity: We present the similarity scores obtained between many selected word pairs, and compare with the scores of other published word vector representations. The comparison of the performances should be done using the judgment of the reader.
2. Human Judged Similarity: We apply our approach and some competing methods on benchmark word pair lists. These lists, published in some papers, have human-judged similarity scores. So this allows comparison with actual numbers.
3. Antonymy Detection: We evaluate our approach on answering closest-opposite questions (extracted from the GRE test), comparing our results with the published ones.
4. Antonym/Synonym Discrimination: We test the performance of our approach in discriminating between antonyms and synonyms. The performance is compared with other published word vector representations.
5. Opinion Mining: We outline the application of our method on an opinion mining task. Moreover, we use other published word vectors to compare the performance on such task.

All used datasets are the same for all the compared models in order to have fair comparisons. In all the conducted experiments, we used the word vectors generated using our implemented algorithm. The training vocabulary used is the one gathered as explained in the previous section. The generated word vector's number of dimensions is 50.

5.1. Word Similarity

Table 2 contains a number of word pairs where the corresponding similarity scores are obtained by the proposed word embedding. The similarity score is computed as the dot product between unit vectors. We have compared our scores with the scores of other published pre-trained word vectors: word2vec [17], GloVe [19], and fastText [20]. The number of dimensions of the vectors in word2vec, GloVe, and fastText is 300, while the dimension of the vectors in our approach is 50. The reason for selecting a number of dimensions of 50 is that more dimensions lead to sparser space, and more overfitting. N/A means that one or both of the words in the given pair have no pre-trained vectors in the method considered. In the table we have included word pairs from different categories:

- Synonyms that occur in the training vocabulary of the proposed method.
- Synonyms that are not part of the training vocabulary of the proposed method (marked with the * symbol in the table).
- Antonyms that are part of the training vocabulary.
- Antonyms that are not included as part of the training vocabulary (also marked with the * symbol).
- Unrelated words that occur in the training vocabulary.
- Unrelated words that are not part of the training vocabulary (marked with the * symbol).
- Word pairs that belong to a certain category (e.g., countries).

Note that the pairs that are not part of the training set constitute an unbiased out of sample test, since the algorithm has not seen them while training.

From Table 2, we can observe several interesting facts:

1. By human judgment, the proposed algorithm seems to be more successful in capturing the similarity between the different words. For example, the pairs "happy–joyful", "amusing–happy" and "district–county" are close synonyms, and our algorithm manages to assign a high similarity. Opposing algorithms give low similarity scores around 0.5 or so.
2. For the antonyms, our algorithm assigns rightly negative numbers. For example, the pairs "modern–outdated" and "unlike–same" are assigned similarities lower than -0.8. The competing algorithms assign similarities in the range of about 0.3 to 0.68, not really signaling that these words are antonyms.
3. For unrelated words, the proposed embedding generally gives them similarities close to zero. For example, the pairs "array–again", "useful–wash" and "decent–morning" are given respectively 0.294, 0.195, and 0.027, which are reasonably close to zero.
4. The competing algorithms are not very successful in differentiating antonyms from unrelated pairs. They give them all comparable scores. For example, the antonyms "modern–outdated" and "unlike–same", and the unrelated pairs "array–again", "useful–wash" and "decent–morning" are given similarities in the same range. It is not clear from the scores whether the pair is an an antonym or an unrelated pair. Additionally the pair "yes–no" is the most basic antonym, and in spite of that, the competing algorithms do not give them a zero score.
5. An antonymous pair such as "happy–unhappy" is given a high similarity score in all the other three competing approaches. Thus it is treated the same as a synonymous pair while it is given a negative similarity score in our approach.
6. An antonymous pair such as "decelerate–speed" is given a low similarity score in all the other three approaches, and thus is treated the same as an unrelated pair while it is given a negative similarity score in our approach.

7. In all the four approaches, the "country–capital" pairs have close similarities.
8. In all the four approaches, the "country–nationality" pairs have close similarities.
9. The pair "Japan–Greece" has a lower similarity score than the pair "Japan–China", although both pairs are country pairs. This is due to the fact that Japan and China have more properties in common (both are Asian countries, and are distance-wise close to each other).
10. A combination such as "African-country" is not found in any of the other three vocabularies. Since the other approaches do not represent such composite words as vectors.
11. We must caution, however, that the competing methods word2vec, GloVe, and fastText are not specifically designed to handle antonyms. Therefore, the comparison presented, which shows clear domination of our method for antonyms, may not be fully due to a particular design or algorithmic outperformance, but also partly due to the fact that the competing methods were not designed to deal with antonyms.

Table 2. Word similarity comparison (* means the pair is not part of the training set for our method).

Word1	Word2	Sphere (Our Approach)	Word2vec [17]	GloVe [19]	FastText [20]
happy	joyful	0.944	0.424	0.598	0.701
amusing	happy	* 0.778	0.27	0.387	0.5
king	queen	* 0.83	0.651	0.76	0.764
boy	girl	* 0.859	0.854	0.825	0.862
male	female	* 0.702	0.841	0.938	0.945
education	student	0.608	0.401	0.682	0.547
movement	motion	0.887	0.231	0.56	0.583
oscillate	move	* 0.828	0.239	0.166	0.354
district	county	* 0.733	0.594	0.651	0.586
district	commune	0.761	0.164	0.367	0.464
friend	partner	0.96	0.37	0.609	0.619
valuable	worthy	0.849	0.351	0.537	0.649
great	tremendous	0.858	0.775	0.668	0.804
joyfulness	joy	* 0.986	0.513	0.362	N/A
France	Paris	0.94	0.555	0.721	0.632
Germany	Berlin	0.949	0.554	0.685	0.673
Egypt	Cairo	0.934	0.602	0.699	N/A
France	French	0.82	0.633	0.686	0.68
Germany	German	0.801	0.681	0.7	0.742
Egypt	Egyptian	0.869	0.602	0.718	0.753
Japan	China	* 0.738	0.315	0.697	0.645
Japan	Greece	* 0.609	0.545	0.494	0.595
african-country	european-country	* 0.88	N/A	N/A	N/A
modern	ongoing	* 0.487	0.129	0.357	0.424
obsolete	outdated	* 0.915	0.743	0.725	0.797
money	shopping	* 0.599	0.166	0.487	0.407
internet	web	0.888	0.597	0.737	0.683
apple	orange	* 0.93	0.392	0.493	0.561
soccer	football	0.926	0.731	0.819	0.804
lion	tiger	* 0.987	0.512	0.621	0.67
giraffe	animal	0.954	0.422	0.436	0.573
light	sun	* 0.612	0.401	0.525	0.558
geology	earth	* 0.701	0.264	0.444	0.531
tenor	opera	* 0.847	0.459	0.395	0.519
unauthorized	simultaneously	* −0.101	0.175	0.237	0.3
bag	country	* 0.061	0.085	0.301	0.367
beach	truck	* 0.448	0.09	0.296	0.421
array	again	* 0.294	-0.011	0.314	0.317
useful	wash	* 0.195	0.073	0.288	0.224
tourism	beach	* 0.513	0.266	0.389	0.459
decent	morning	* 0.027	0.041	0.373	0.253
go	new	* 0.111	0.113	0.564	0.345
yes	no	−1	0.392	0.695	0.549

Table 2. *Cont.*

Word1	Word2	Sphere (Our Approach)	Word2vec [17]	GloVe [19]	FastText [20]
active	passive	−0.916	0.436	0.577	0.678
happy	unhappy	−0.912	0.613	0.565	0.767
young	old	−0.841	0.417	0.573	0.583
youth	old	* −0.357	0.212	0.384	0.392
youth	elderly	−0.674	0.215	0.39	0.499
valuable	valueless	* −0.912	0.334	0.262	0.614
recall	forget	* −0.907	0.3	0.532	0.567
education	ignorance	−0.854	0.2	0.306	0.487
complex	easy	* −0.769	0.213	0.439	0.453
beginning	deadline	* −0.73	0.22	0.447	0.348
modern	outdated	* −0.933	0.404	0.427	0.528
unlike	same	* −0.957	0.299	0.681	0.482
grow	eradicate	* −0.858	0.241	0.308	0.491
certainty	uncertainty	−0.931	0.492	0.617	0.706
truth	lying	* −0.766	0.237	0.468	0.479
daring	coward	* −0.676	0.196	0.233	0.432
decelerate	speed	* −0.843	0.212	0.22	0.385

5.2. Human Judged Similarity

We considered here datasets from other researchers that attached human-judged similarity scores to the word pairs. This would give an unbiased assessment, as the similarity estimate is performed by different researchers. None of the human-judged similarity scores associated with the pairs in these datasets are included in our training set, in order to have it as an out of sample test.

We considered the labeled data of MC30 [56] and RG [57], which have human-judged similarity measures. There is a third dataset available, namely, the WordSim-353 [58] human-judged dataset, but we did not perform a test using this dataset because they estimated antonyms as similar while in our approach we consider antonyms as opposites. We computed two metrics, the Spearman rank correlation (Sp) and root mean square error (RMSE). Moreover, we have compared the obtained metrics' values with those of word2vec [17], GloVe [19], and fastText [20]. Table 3 shows that we achieve the best results for both metrics on both datasets.

Table 3. Results of human-judged datasets.

Approach	MC30		RG	
	Sp	**RMSE**	**Sp**	**RMSE**
Sphere (our approach)	**0.91**	**0.18**	**0.9**	**0.15**
word2vec	0.8	0.3	0.76	0.29
GloVe	0.79	0.27	0.82	0.25
fastText	0.83	0.24	0.84	0.22

5.3. Antonymy Detection

We have applied our approach on answering the closest-opposite GRE questions, collected in [27,30]. The GRE, or Graduate Record Examinations, is a worldwide exam needed for admission to graduate schools. The verbal part has a group of multiple choice questions that seek the closest-opposite. The 162 questions of the development set are used as a part of our training vocabulary; i.e., the words together with the correct answers are added to the training vocabulary as antonymous pairs. On the other hand, both the 950 questions and the 790 questions datasets are used as test sets (i.e., their pairs do not exist in our training set).

We applied our word embedding method to compute the similarities between the word in question and all candidate answers, and selected the answer that is closest to −1, signifying an

antonym. We have computed the precision, recall, and F-score as given in [27], so that we could make comparisons with the numbers given in the competing methods. They are given by

$$Precision(P) = \frac{\text{Number of questions answered correctly}}{\text{Number of questions attempted}} \tag{14}$$

$$Recall(R) = \frac{\text{Number of questions answered correctly}}{\text{Total number of questions}} \tag{15}$$

$$F\text{-}score = \frac{2 \times P \times R}{P + R} \tag{16}$$

In our case we note that the precision = recall = F-score; this is because our approach attempted all the questions. All the words that exist in the questions and in the candidate answers have corresponding vectors in our word embedding method.

Furthermore, we compared our results with the best results recorded in [27,29–32,34] that use the same dataset. From Table 4, it is shown that we achieved the best precision, recall, and F-score for the development set. We achieved the second best scores in both test sets after [34]; however, the dimension of their vectors is 300 while the dimension of our vectors is only 50. As mentioned earlier, their model is trained such that there is no differentiation between antonymous pairs and unrelated pairs, as both will have low similarity scores. Moreover, we were able to compare our results with [34] only on antonymy detection as the authors did not include the performance of their vectors on synonyms or unrelated words.

Table 4. Results of closest-opposite GRE.

Approach	Development Set			Test Set 950			Test Set 790		
	Precision	Recall	F-Score	Precision	Recall	F-Score	Precision	Recall	F-Score
Sphere (our approach)	**0.98**	**0.98**	**0.98**	0.83	0.83	0.83	0.8	0.8	0.8
Mohammad et al., 2008 [27]	0.76	0.66	0.70	0.76	0.64	0.70	-	-	-
Yih et al., 2012 [29]	0.88	0.87	0.87	0.81	0.80	0.81	-	-	-
Mohammad et al., 2013 [30]	0.79	0.66	0.72	-	-	-	0.77	0.63	0.69
Chang et al., 2013 [31]	0.88	0.85	0.87	0.81	0.77	0.79	-	-	-
Zhang et al., 2014 [32]	0.88	0.88	0.88	0.82	0.82	0.82	-	-	-
Ono et al., 2015 [34]	0.92	0.91	0.91	**0.90**	**0.88**	**0.89**	**0.89**	**0.87**	**0.88**

5.4. Antonym/Synonym Discrimination

In this section, we show using statistical measures how our approach perform in the task of discriminating between antonyms and synonyms. Moreover, our approach's performance is compared to the other published word vectors' models: word2vec [17], GloVe [19], and fastText [20]. The used dataset is introduced by [35]. This dataset considers word pairs in three categories: adjectives, nouns and verbs. The pairs are marked as antonyms or synonyms. Therefore, we have conducted this experiment as a binary classification task with two classes namely, antonyms and synonyms. The classification is done based on the similarity score between the vectors of the word pair. If this similarity score is equal to or greater than a certain threshold (more specifically 0.5), then the pair is classified as synonyms otherwise the pair is classified as antonyms. The used dataset is refined such that pairs that have any non-existent word in any of the models are removed. After this refinement the dataset has the following structure:

- 470 adjectives: (238 antonyms and 232 synonyms);
- 547 nouns: (276 antonyms and 271 synonyms);

- 632 verbs: (311 antonyms and 321 synonyms).

In Tables 5–7 the performance measures are recorded for the four models and for the three categories respectively. The results show that our model outperforms all the other three models in all the categories.

Table 5. Antonym/synonym discrimination results—adjectives.

Approach	Precision	Recall	F-Score	Accuracy
Sphere (our approach)	**0.86**	**0.83**	**0.83**	**82.98%**
word2vec	0.61	0.56	0.50	55.96%
GloVe	0.65	0.60	0.57	60.43%
fastText	0.67	0.67	0.67	66.81%

Table 6. Antonym/synonym discrimination results—nouns.

Approach	Precision	Recall	F-Score	Accuracy
Sphere (our approach)	**0.72**	**0.72**	**0.72**	**72.39%**
word2vec	0.65	0.58	0.51	57.77%
GloVe	0.67	0.62	0.58	61.61%
fastText	0.67	0.66	0.65	65.81%

Table 7. Antonym/synonym discrimination results—verbs.

Approach	Precision	Recall	F-Score	Accuracy
Sphere (our approach)	**0.83**	**0.81**	**0.81**	**81.49%**
word2vec	0.70	0.54	0.43	53.80%
GloVe	0.69	0.59	0.53	59.02%
fastText	0.66	0.63	0.61	62.66%

5.5. Comments on the Results

We can observe that the proposed word embedding approach gives more reasonable similarity scores than some of the major approaches, such as word2vec, GloVe, and fastText. These methods have a particular deficit dealing with antonyms, and distinguishing between antonyms and unrelated words. The failure of some of them in assigning the right similarity score to the pair "yes" and "no" is case in point. Our algorithm also fared better on the two benchmarks tested. It also did well compared to other algorithms that are specifically designed to deal with antonyms, on the GRE antonym dataset. Furthermore, our approach proved its efficiency in discriminating between antonyms and synonyms as compared to other published word vectors' models. Our algorithm can handle composite words (like "fairy tale"). It could potentially also handle words with multiple meanings, such as "bat" (the animal) and "bat" (a club). The way to tackle these is to consider them as different words, like "bat-1" and "bat-2". The challenge facing all word embedding methods is to distinguish words with multiple meanings from the context of the sentence.

5.6. Opinion Mining

Opinion mining application refers to classifying a review as positive or negative. We applied our approach to learn vectors for words that have bias from which sentiments can be inferred. We have begun with a small set of such words then grow our sphere by obtaining synonyms and antonyms for these words. The generated word vectors are used to compute the sentiment that the review reflects. Every word in the review has two sub-scores that are obtained by respectively the dot product between this word and a number of positive words on one hand, and a number of negative words

on the other hand, which are specifically collected to gauge sentiment. We applied the approach to 20,000 hotel reviews [59] from "515K Hotel Reviews Data in Europe" dataset [60] that are collected from booking.com where each review is annotated as positive or negative. Furthermore, we used vectors from other models to compare the results. We achieved the highest F-score compared to word2vec [17], GloVe [19], and fastText [20]. The scores obtained are 0.81, 0.79, 0.74, and 0.79 respectively [61].

6. Conclusions

In this work we have devised a new approach for embedding words into a sphere. The algorithm is a simple relaxation one without the need for intensive training phases. The approach is a polarity capturing one in the sense that antonymous word pairs are located at opposite poles of the sphere. Not only antonymy can be detected but also other relations (e.g. unrelated pairs).

We have managed to gather an adequate vocabulary that can flexibly be extended. Moreover, we evaluated our approach using several datasets, showing that the approach competes well with other approaches in the literature. We have successfully proved that our approach does not suffer from the ambiguity in differentiating between various word pair relations such as synonyms, antonyms, and unrelated pairs.

Author Contributions: Conceptualization, S.R. and A.F.A.; methodology, S.R., A.F.A. and S.S.; software, S.R.; validation, S.R.; formal analysis, S.R., A.F.A., and S.S.; investigation, S.R., A.F.A., and S.S.; resources, S.R.; data curation, S.R.; writing—original draft preparation, S.R.; writing—review and editing, S.R. and A.F.A.; visualization, S.R.; supervision, A.F.A. and S.S.; project administration, A.F.A. and S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to acknowledge the help and the discussions with Professor Aly Aly Fahmy of the Arab Academy of Science and Technology, and Professor William Hager of University of Florida.

Conflicts of Interest: The authors declare no conflict of interest.

Data Availability: The word vectors generated are found at https://github.com/SandraRizkallah/Sphere-Eng-WordVectors.

Appendix A. Proof of Theorem 1: Convergence of the Algorithm

When we consider a particular word, we apply Equations (9) and (11), in order to tune the position of the word's designated vector. To prove that the repeated application of this step (for one word after another) leads to convergence, we show that each application leads to a non-increase of the error function Equation (1). Since this error function is bounded from below by zero, the algorithm converges to some limit of the error function.

This formulation of an optimization falls under the form of optimization of a quadratic function on the sphere, i.e., subject to the solution lying on the sphere $\|x\|^2 = r^2$. It has been studied in detail by [62–65], as it is applied in types of optimization methods called trust region methods of nonlinear optimization.

Let the errors before and after applying (9) be E_1 and E_2 respectively (we mean the errors given by Equation (1)). Since both before and after the update the vector x_k would be normalized to become unit length, we can use V (given in (3), (4)) in place of E. The subtracted term $\lambda\left({x_k}^T x_k - 1\right)$ will equal zero in both cases, so it will not impact it. Therefore

$$\begin{aligned} E_2 - E_1 &= V_2 - V_1 \\ &= {x_k}^T A x_k - 2b^T x_k - \left[{x'_k}^T A x'_k - 2b^T x'_k\right] \end{aligned} \tag{A1}$$

where x'_k is the word vector before the update. After applying Equation (9) it becomes x_k, given by:

$$\begin{aligned} x_k &= A^{-1} b \\ &= \left(A' - \lambda I\right)^{-1} b \end{aligned} \tag{A2}$$

where the last step follows from the definitions of A and A' (5) and (12). As mentioned, λ is determined from:

$$b^T \left[A' - \lambda I\right]^{-2} b = 1 \tag{A3}$$

According to [62–65] Equation (A3) has at most $2N$ solutions for λ, where N is the dimension of the vector x_k, with the smallest and largest solutions corresponding to respectively the minimum and the maximum of the optimization problem. The solutions in the middle correspond to the saddle points. The smallest solution obeys the condition $\lambda \leq$ all eigenvalues of A' [64], which means that $A' - \lambda I$ is positive semi-definite (because the eigenvalues of $A' - \lambda I$ equal the eigenvalues of A' minus λ). Because of these conditions, the aforementioned works of [62–65] prove that $E_2 - E_1 \leq 0$.

Irrespective of their proof, we formulated a simplified proof, given as follows: Substituting (A2) into (A1), we get

$$\begin{aligned} E_2 - E_1 &= b^T A^{-1} A A^{-1} b - 2b^T A^{-1} b - \left[x_k'^T A x_k' - 2b^T x_k'\right] \\ &= -b^T A^{-1} b - \left[x_k'^T A x_k' - 2b^T x_k'\right] \\ &= -\left[Ax_k' - b\right]^T A^{-1} \left[Ax_k' - b\right] \end{aligned} \tag{A4}$$

The latter line follows by expanding the two multiplied brackets. We find that it equals the expression in the preceding line. Since the matrix $A = A' - \lambda I$, and the latter is shown to be positive semi-definite,

$$E_2 - E_1 \leq 0 \tag{A5}$$

because any quadratic form $y^T Q y \geq 0$ for any positive semi-definite matrix Q and any vector y. So the error reduces after each iteration, or stays the same. After applying the update for every word, and having a complete cycle where the error does not reduce any more for any of the vectors, convergence of the algorithm is signified.

References

1. Zou, W.Y.; Socher, R.; Cer, D.; Manning, C.D. Bilingual word embeddings for phrase-based machine translation. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, WA, USA, 18–21 October 2013; pp. 1393–1398.
2. Maas, A.L.; Daly, R.E.; Pham, P.T.; Huang, D.; Ng, A.Y.; Potts, C. Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, Portland, OR, USA, 19–24 June 2011; pp. 142–150.
3. Tang, D.; Wei, F.; Yang, N.; Zhou, M.; Liu, T.; Qin, B. Learning sentiment-specific word embedding for twitter sentiment classification. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Baltimore, MD, USA, 22–27 June 2014; Volume 1, pp. 1555–1565.
4. Dragoni, M.; Petrucci, G. A neural word embeddings approach for multi-domain sentiment analysis. *IEEE Trans. Affect. Comput.* **2017**, *8*, 457–470. [CrossRef]
5. Deho, B.O.; Agangiba, A.W.; Aryeh, L.F.; Ansah, A.J. Sentiment Analysis with Word Embedding. In Proceedings of the 2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST), University of Ghana, Legon, Accra, Ghana, 22–24 August 2018; pp. 1–4.
6. Liang, J.; Cui, B.; Jiang, H.; Shen, Y.; Xie, Y. Sentence similarity computing based on word2vec and LSTM and its application in rice FAQ question-answering system. *J. Nanjing Agric. Univ.* **2018**, *41*, 946–953.
7. Liu, H. Agricultural Q&A System Based on LSTM-CNN and Word2vec. *Revis. Fac. Agron. Univ. Zulia* **2019**, *36*, 543–551.
8. Roy, D. Word Embedding based Approaches for Information Retrieval. In Proceedings of the Seventh BCS-IRSG Symposium on Future Directions in Information Access 7, Barcelona, Spain, 5 September 2017; pp. 1–4.

9. Liu, Y.; Fu, Z. Secure search service based on word2vec in the public cloud. *Int. J. Comput. Sci. Eng.* **2019**, *18*, 305–313. [CrossRef]
10. Kim, J.; Hong, T.; Kim, P. Word2Vec based spelling correction method of Twitter message. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; pp. 2016–2019.
11. Pan, Q.; Dong, H.; Wang, Y.; Cai, Z.; Zhang, L. Recommendation of Crowdsourcing Tasks Based on Word2vec Semantic Tags. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 2121850. [CrossRef]
12. Habibi, M.; Weber, L.; Neves, M.; Wiegandt, D.L.; Leser, U. Deep learning with word embeddings improves biomedical named entity recognition. *Bioinformatics* **2017**, *33*, i37–i48. [CrossRef]
13. Kågebäck, M.; Mogren, O.; Tahmasebi, N.; Dubhashi, D. Extractive summarization using continuous vector space models. In Proceedings of the 2nd Workshop on Continuous Vector Space Models and Their Compositionality (CVSC), Gothenburg, Sweden, 26–30 April 2014; pp. 31–39.
14. Rossiello, G.; Basile, P.; Semeraro, G. Centroid-based text summarization through compositionality of word embeddings. In Proceedings of the MultiLing 2017 Workshop on Summarization and Summary Evaluation Across Source Types and Genres, Valencia, Spain, 3 April 2017; pp. 12–21.
15. Yang, K.; Al-Sabahi, K.; Xiang, Y.; Zhang, Z. An integrated graph model for document summarization. *Information* **2018**, *9*, 232. [CrossRef]
16. Simard, P.Y.; LeCun, Y.A.; Denker, J.S.; Victorri, B. Transformation invariance in pattern recognition—Tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*; Springer: New York, NY, USA, 1998; pp. 239–274.
17. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. *Adv. Neural Inf. Process. Syst. 26 (NIPS 2013)* **2013**, 3111–3119.
18. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arXiv:1301.3781.
19. Pennington, J.; Socher, R.; Manning, C. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
20. Mikolov, T.; Grave, E.; Bojanowski, P.; Puhrsch, C.; Joulin, A. Advances in pre-training distributed word representations. *arXiv* **2017**, arXiv:1712.09405.
21. Dev, S.; Hassan, S.; Phillips, J.M. Absolute Orientation for Word Embedding Alignment. *arXiv* **2018**, arXiv:1806.01330.
22. Vilnis, L.; McCallum, A. Word representations via gaussian embedding. *arXiv* **2014**, arXiv:1412.6623.
23. Bian, J.; Gao, B.; Liu, T.Y. Knowledge-powered deep learning for word embedding. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*; Springer: New York, NY, USA, 2014; pp. 132–148.
24. Zhou, G.; He, T.; Zhao, J.; Hu, P. Learning continuous word embedding with metadata for question retrieval in community question answering. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Beijing, China, 26–31 July 2015; Volume 1, pp. 250–259.
25. Faruqui, M.; Dodge, J.; Jauhar, S.K.; Dyer, C.; Hovy, E.; Smith, N.A. Retrofitting word vectors to semantic lexicons. *arXiv* **2014**, arXiv:1411.4166.
26. Jo, H. Expansional Retrofitting for Word Vector Enrichment. *arXiv* **2018**, arXiv:1808.07337.
27. Mohammad, S.; Dorr, B.; Hirst, G. Computing word-pair antonymy. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2008; pp. 982–991.
28. Lobanova, A. *The Anatomy of Antonymy: A Corpus-Driven Approach*; University of Groningen: Groningen, The Netherlands, 2012.
29. Yih, W.t.; Zweig, G.; Platt, J.C. Polarity inducing latent semantic analysis. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2012; pp. 1212–1222.
30. Mohammad, S.M.; Dorr, B.J.; Hirst, G.; Turney, P.D. Computing lexical contrast. *Comput. Linguist.* **2013**, *39*, 555–590. [CrossRef]

31. Chang, K.W.; Yih, W.t.; Meek, C. Multi-relational latent semantic analysis. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, WA, USA, 18–21 October 2013; pp. 1602–1612.
32. Zhang, J.; Salwen, J.; Glass, M.; Gliozzo, A. Word semantic representations using bayesian probabilistic tensor factorization. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1522–1531.
33. Santus, E.; Lu, Q.; Lenci, A.; Huang, C.R. Taking antonymy mask off in vector space. In Proceedings of the 28th Pacific Asia Conference on Language, Information and Computing, Phuket, Thailand, 12–14 December 2014; pp. 135–144.
34. Ono, M.; Miwa, M.; Sasaki, Y. Word embedding-based antonym detection using thesauri and distributional information. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, CO, USA, 31 May–5 June 2015; pp. 984–989.
35. Nguyen, K.A.; im Walde, S.S.; Vu, N.T. Integrating Distributional Lexical Contrast into Word Embeddings for Antonym-Synonym Distinction. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Berlin, Germany, 7–12 August 2016.
36. Li, L.; Qin, B.; Liu, T. Contradiction detection with contradiction-specific word embedding. *Algorithms* **2017**, *10*, 59. [CrossRef]
37. Zahran, M.A.; Magooda, A.; Mahgoub, A.Y.; Raafat, H.; Rashwan, M.; Atyia, A. Word representations in vector space and their applications for arabic. In *International Conference on Intelligent Text Processing and Computational Linguistics*; Springer: New York, NY, USA, 2015; pp. 430–443.
38. Soliman, A.B.; Eissa, K.; El-Beltagy, S.R. Aravec: A set of arabic word embedding models for use in arabic nlp. *Procedia Comput. Sci.* **2017**, *117*, 256–265. [CrossRef]
39. Mihalcea, R.; Radev, D. *Graph-Based Natural Language Processing and Information Retrieval*; Cambridge University Press: Cambridge, UK, 2011.
40. Venkatesh, S.S. *The Theory of Probability: Explorations and Applications*; Cambridge University Press: Cambridge, UK, 2013.
41. NetworkX Developers. 2015. Available online: https://networkx.github.io/documentation/networkx-1.10/ (accessed on 4 July 2019).
42. Smart Words—A Handpicked Collection of Gems of the English Language. Available online: http://www.smart-words.org/list-of-synonyms/ (accessed on 4 July 2019).
43. Power Thesaurus. Available online: https://www.powerthesaurus.org (accessed on 4 July 2019).
44. Princeton University "About WordNet". 2010. Available online: https://wordnet.princeton.edu/ (accessed on 4 July 2019).
45. 100 Examples of Antonyms. Available online: https://www.powerthesaurus.org/100_examples_of_antonyms (accessed on 23 July 2018).
46. List 24-Synonyms. Available online: http://myenglishgrammar.com/list-24-synonyms.html (accessed on 23 July 2019).
47. Course Hero.docx-SYNONYMS. Available online: https://www.coursehero.com/file/38484777/course-herodocx/ (accessed on 23 July 2019).
48. Synonyms for the 96 Most Commonly Used Words in English. Available online: https://justenglish.me/2014/04/18/synonyms-for-the-96-most-commonly-used-words-in-english/ (accessed on 23 July 2019).
49. List 23-Antonyms. Available online: http://myenglishgrammar.com/list-23-antonyms.html (accessed on 23 July 2019).
50. Fry, E.B.; Kress, J.E. *The Reading Teacher's Book of Lists*; John Wiley & Sons: New York, NY, USA, 2012; Volume 55.
51. List of 30 Antonyms You Should Know. Available online: https://www.indiatoday.in/education-today/grammar-vocabulary/story/antonyms-264084-2015-09-21 (accessed on 23 July 2018).
52. Common Opposites-Antonyms Vocabulary Word List. Available online: https://www.enchantedlearning.com/wordlist/opposites.shtml (accessed on 23 July 2018).
53. Antonym Word List. Available online: http://slplessonplans.com/files/antonymlist.pdf (accessed on 23 July 2018).

54. Thesaurus.com. The World's Favorite Online Thesaurus! 2013. Available online: https://www.thesaurus.com/ (accessed on 4 July 2019).
55. Michael Fogleman: Random Phrases. Available online: https://www.michaelfogleman.com/phrases/ (accessed on 4 July 2019).
56. Miller, G.A.; Charles, W.G. Contextual correlates of semantic similarity. *Lang. Cogn. Process.* **1991**, *6*, 1–28. [CrossRef]
57. Rubenstein, H.; Goodenough, J.B. Contextual correlates of synonymy. *Commun. ACM* **1965**, *8*, 627–633. [CrossRef]
58. Finkelstein, L.; Gabrilovich, E.; Matias, Y.; Rivlin, E.; Solan, Z.; Wolfman, G.; Ruppin, E. Placing search in context: The concept revisited. *ACM Trans. Inf. Syst.* **2002**, *20*, 116–131.
59. Li, Q.; Li, S.; Zhang, S.; Hu, J.; Hu, J. A Review of Text Corpus-Based Tourism Big Data Mining. *Appl. Sci.* **2019**, *9*, 3300. [CrossRef]
60. Liu, J. [dataset] 515K Hotel Reviews Data in Europe. Available online: https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe (accessed on 12 October 2019).
61. Rizkallah, S.; Atiya, A.; Shaheen, S. Learning Spherical Word Vectors for Opinion Mining and Applying on Hotel Reviews. *Work. Pap.* **2020**.
62. Busygin, S. A new trust region technique for the maximum weight clique problem. *Discret. Appl. Math.* **2006**, *154*, 2080–2096. [CrossRef]
63. Busygin, S.; Butenko, S.; Pardalos, P.M. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. *J. Comb. Optim.* **2002**, *6*, 287–297. [CrossRef]
64. Hager, W.W. Minimizing a quadratic over a sphere. *Siam J. Optim.* **2001**, *12*, 188–208. [CrossRef]
65. Forsythe, G.E.; Golub, G.H. On the stationary values of a second-degree polynomial on the unit sphere. *J. Soc. Ind. Appl. Math.* **1965**, *13*, 1050–1068. [CrossRef]