A Method for Improving Word Representation Using Synonym Information

Huyen Trang Phan[1][0000−0002−7466−9562], Ngoc Thanh Nguyen[2][0000−0002−3247−2948], Javokhir Musaev[1][0000−0003−4656−0479], and Dosam Hwang[1,⋆][0000−0001−7851−7323]

[1] Department of Computer Engineering, Yeungnam University, Gyeongsan, South Korea
huyentrangtin@gmail.com, javokhirmuso@yu.ac.kr, dshwang@yu.ac.kr
[2] Department of Applied Informatics, Wroclaw University of Science and Technology, Wroclaw, Poland
Ngoc-Thanh.Nguyen@pwr.edu.pl

Abstract. The emergence of word embeddings has created good conditions for natural language processing used in an increasing number of applications related to machine translation and language understanding. Several word-embedding models have been developed and applied, achieving considerably good performance. In addition, several enriching word embedding methods have been provided by handling various information such as polysemous, subwords, temporal, and spatial. However, prior popular vector representations of words ignored the knowledge of synonyms. This is a drawback, particularly for languages with large vocabularies and numerous synonym words. In this study, we introduce an approach to enrich the vector representation of words by considering the synonym information based on the vectors' extraction and presentation from their context words. Our proposal includes three main steps: First, the context words of the synonym candidates are extracted using a context window to scan the entire corpus; second, these context words are grouped into small clusters using the latent Dirichlet allocation method; and finally, synonyms are extracted and converted into vectors from the synonym candidates based on their context words. In comparison to recent word representation methods, we demonstrate that our proposal achieves considerably good performance in terms of word similarity.

Keywords: Synonym words · Word embeddings · Synonym vector.

1 Introduction

Embeddings, similar to vector models that capture relational meaning, are more fine-grained than just a string or index; in particular, embeddings are good at modeling similarities/analogies. To apply embeddings, we only need to download and use them. They are useful tools in practice and are more popular in several

⋆ Corresponding author

fields, specifically, word embeddings in the natural language processing area. Word embeddings represent word meanings from corpus statistics.

The use of word embeddings has been widely increasing in applications related to natural language processing, such as sequence tagging [15], machine translation [21], language understanding [18], text classification [13] and sentiment analysis [20]. In addition, word embeddings are useful in machine learning and deep learning algorithms. Therefore, several pre-trained word embeddings exhibit state-of-the-art performance. These methods are applied in several studies, such as Word2Vec [16], FastText [12], GloVe [17], and BERT [4]. Word2Vec[3] includes two models: skip-gram and continuous bag-of-words. The first model predicts the surrounding words for the current word, meanwhile the second model uses the context words to predict the current word [28, 29]. This model produces the same vector for a word irrespective of its meaning and context. GloVe[4], used for word representations, leverages the statistics of word occurrences in the corpus and uses a neural network to represent the meaning of such statistics [28, 29]. The idea of this model is similar to that of latent semantic analysis. It captures the global and local contexts of the word. FastText[5] is improved from the Word2Vec skip-gram model by considering the subword information. A word is represented as a sum of character n-gram embeddings that appeared in the word. The FastText model outperforms skip-gram model in most scenarios and datasets when dealing with syntactic tasks [29]. However, for semantic tasks, the FastText model is less accurate than the skip-gram model [29]. It can generate out-of-vocabulary word embeddings. BERT[6] combines several tasks. It predicts masked words in a sentence and indicates whether sentence A is followed by sentence B, as embedding combines several hidden layers of the network [28, 29]. In addition, BERT learns relationships between sentences and predicts whether sentence B is the actual sentence that follows sentence A or whether it is a random sentence. However, the above methods have the same limitation that ignores the impact of synonyms when representing words [28].

According to WordNet, synonym words are introduced as *"words that denote the same concept and are interchangeable in many contexts"*. To clearly understand the importance of synonyms in the tweet sentiment analysis task, we consider the following small example. Assume that there are two tweets as follows: Tweet 1: *"The color of this phone is outdated"*. Tweet 2: *"The color of this phone is outmoded"*. It can be seen that the words "outdated" and "outmoded" have the same meaning, but they have entirely different characters in words. These words are called synonyms. However, most of the previous methods used different vectors to represent them. This leads to a misunderstanding of the word meaning by the computer. Therefore, the quality of word representations is decreased, and the performance of applications is also affected. To address the aforementioned synonyms problem, we introduce a model to enrich

[3] https://code.google.com/archive/p/word2vec/
[4] https://nlp.stanford.edu/projects/glove/
[5] https://fasttext.cc/docs/en/crawl-vectors.html
[6] https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/

the vector representation of words by adding the synonym information. This proposal focuses on the extraction and presentation of synonyms based on their context words by considering three main steps: First, the context words of the synonym candidates are extracted; second, these context words are grouped into small clusters using the latent Dirichlet allocation (LDA) method; and finally, synonyms are extracted and converted into vectors from the synonym candidates based on their context words. In comparison to recent word representation methods, we demonstrate that our proposed method achieves state-of-the-art performance on a given task in terms of word similarity. Our proposal is motivated by the distributional hypothesis [9] that says: *"words that occur in the same contexts tend to have similar meanings"* and the basic hypothesis investigated by Rubenstein *et al.* [24]: *"there is a positive relationship between the degree of synonymy (semantic similarity) existing between a pair of words and the degree to which their contexts are similar"*.

The remainder of this paper is organized as follows: The literature regarding sentiment analysis methods is summarized in Section 2. We describe the research problem in Section 3 and introduce the proposed method in Section 4. The information related to experimental results, such as data acquisition, evaluation method, and result discussion, is provided in Section 5. The conclusions and future work are discussed in the final section.

2 Related Works

Recently, several methods have been published to enrich word embeddings. In this section, we represent certain recent and outstanding methods by discussing their processes, advantages, and disadvantages.

Svoboda *et al.* introduced two approaches to improve the quality of vector representations. In [27], the authors enriched word embeddings by considering global information. In [26], the authors improved word meaning representations using Wikipedia categories. Jianqiang *et al.* [11] provided enriching word embedding approaches by using the word vectors in the GloVe collection, n-grams of words, and the sentiment score of words. The model obtained good results. However, the authors did not compare the performance with other studies on the same datasets. Meanwhile, Hassan *et al.* [14] converted words into feature vectors of real values. These features include the semantic and syntactic information. However, this method only considered the word's surface features ignoring the impact of the in-depth features. Therefore, in [1], the authors improved word embeddings by adding the information related to the features, such as generic words and sentiment specific words. Rezaeinia *et al.* [22] increased the performance of available word embeddings by considering the information of words, such as the part-of-speech (POS) tag, lexicon, and position. Nevertheless, the authors ignored the subword information, global information, and temporal and spatial information. Therefore, Bojanowski *et al.* [3] proposed a new approach based on the skip-gram model. Each word was represented as a bag of character n-grams to overcome the limitation of ignoring the morphology of words. A vector representation was associated with each character n-gram, and words were represented as the sum of these representations. This method could quickly train models on large corpora and compute word representations for words that did not appear in the training data. Besides, Gong *et al.* [6] proved that the meaning of a word is closely linked to sociocultural factors that can change over time and location, resulting in corresponding meaning changes. Therefore, they presented a model for learning word representation conditioned on time and location to solve the problem of ignoring the previous methods' temporal or spatial information. In addition, to capture meaning changes over time and location, the authors required that the resulting word embeddings retained salient semantic and geometric properties. This model was trained on time- and location-stamped corpora and used both quantitative and qualitative evaluations to capture semantics across time and locations. Whatever, the discussed methods did not consider the impact of the polysemous words in word embeddings. To solve this limitation, Gou *et al.* [7] presented an approach to convert the polysemous into vectors by clustering the context words. This method is the basis of our improvement method. The difference is selecting the word embedding model, extracting and clustering context words, and determining the parameters' value to predict the synonyms.

Notably, the prior methods did not consider the impact of synonyms when representing words in the vector space. Therefore, we chose to study this problem.

3 Model

Let $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ represent a set of tweets, where $\mathcal{W}_t = \{w_1, w_2, ..., w_h\}$ represent a set of words existing in tweet t. Let $\mathcal{W} = \{w_1, w_2, ..., w_m\}, (m > h)$ represent a set of words in the vocabulary, where $\mathcal{W} = \cup_{t\in\mathcal{T}}\{\mathcal{W}_t\}$. Let $\mathcal{V} = \{\mathcal{V}_{w_1}, \mathcal{V}_{w_2}, ..., \mathcal{V}_{w_m}\}$, where $\mathcal{V}_{w_j} = \{v^1_{w_j}, v^2_{w_j}, ..., v^q_{w_j}\}$, represent a set of context words that surround the words in set $\mathcal{W}$.

3.1 General Model

In this section, we briefly review the skip-gram model introduced by Mikolov *et al.* [16]. Given a word vocabulary $\mathcal{W}$, the goal of the skip-gram model is to learn a vector representation for each word w. In other words, the aim of this model is to maximize the average log probability as follows:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{q} log\, p(v^j_{w_i}|w_i) \tag{1}$$

The probability of observing a context word $v^j_{w_i}$, given w_i, is parameterized. Let $\mathcal{S}c$ denote a scoring function that maps pairs of (word, context) to scores in $\mathbb{R}$. The problem is to predict context words. For the word w_i, all context words as positive examples and sample negatives from vocabulary are considered [3]. For a target context word $v_{w_i}^j$, using the binary logistic loss, the negative log-likelihood is shown as follows:

$$log(1 + e^{-Sc(w_i, v_{w_i}^j)}) + \sum_{n \in \mathcal{N}_{i,j}} log(1 + e^{Sc(w_i, n)}) \quad (2)$$

where $\mathcal{N}_{i,j}$ represents a set of negative examples sampled from the vocabulary. By denoting the logistic loss function $\ell : x \to log(1 + e^{-x})$, equation 1 is rewritten as follows:

$$\sum_{i=1}^{n} \left[\sum_{j=1}^{k} \ell(Sc(w_i, v_{w_i}^j)) + \sum_{n \in \mathcal{N}_{i,j}} \ell(-Sc(w_i, n)) \right] \quad (3)$$

Assume two vectors z_{w_i} and $z_{v_{w_i}^j}$, corresponding to word w_i and context word $v_{w_i}^j$, respectively. Then the score Sc is computed as the scalar product between word w_i and context word $v_{w_i}^j$ using the following equation:

$$Sc(w_i, v_{w_i}^j) = z_{w_i}^{\top} z_{v_{w_i}^j} \quad (4)$$

3.2 Synonym Representation Model

Given a word $w \in \mathcal{W}$, let $S_w = \{s_w^1, s_w^2, ..., s_w^g\}$ represents a set of synonym words of word w, where each synonym word $s_w^g \in S_w$ is associated to a vector representation $z_{s_w^g}$. A synonym word is represented by the sum of the vector representations of its contexts. Thus, the scoring function of our model is shown as follows:

$$Sc(s_w, v_w) = \sum_{v_w \in \mathcal{V}_w} z_{v_w}^{\top} z_{s_w} \quad (5)$$

Similar to the Word2Vec skip-gram model, our model decides whether the target word is a synonym word or not by using the context words of the target word. In addition, to identify the synonym word, we use a context window of size between one and five words to decide the context words of the synonym candidate word. Next, we formally define the problems related to the enrichment of word embeddings by considering synonym information. As a computational problem, the improvement of word embeddings assumes that the input is a set of tweets $\mathcal{T} = \{t_1, t_2, ..., t_n\}$.

For $t \in \mathcal{T}$: let $\mathcal{W}_t = \{w_1, w_2, ..., w_h\}$ represent a set of words appearing in t. Let $\mathcal{W} = \{w_1, w_2, ..., w_m\}, (m > h)$ represent a set of words in the vocabulary, where $\mathcal{W} = \cup_{t \in \mathcal{T}} \{\mathcal{W}_t\}$. For $w \in \mathcal{W}$: let $\mathcal{U} = \{u_{w_1}, u_{w_2}, ..., u_{w_m}\}$ be a set of baseline word embeddings of words in $\mathcal{W}$, and u_{w_j} $(j = 1, ..., m)$ denote the vector of word w_j. For $w \in \mathcal{W}_t$ and $u_w \in \mathcal{U}$: let $\mathcal{M}$ represent a word embeddings mapping table, $\mathcal{M} = \{[w_1, u_{w_1}], [w_2, u_{w_2}], ..., [w_m, u_{w_m}]\}$. For $w \in \mathcal{W}_t$ and $\mathcal{V}_w \in \mathcal{V}$: let $\mathcal{P}$ represent a context words mapping table of words, $\mathcal{P} = \{[w_1, \mathcal{V}_{w_1}], [w_2, \mathcal{V}_{w_2}], ..., [w_m, \mathcal{V}_{w_m}]\}$. From $\mathcal{P}$ and $\mathcal{M}$: let $\mathcal{G} = \{\mathcal{G}_{w_1}, \mathcal{G}_{w_2}, ..., \mathcal{G}_{w_m}\}$ represent a set of clusters of context words, in which $\mathcal{G}_{w_j} = \{g_{w_j}^1, g_{w_j}^2, ..., g_{w_j}^k\}$ $(j = 1, ..., m)$; $g_{w_j}^i = \{v_{w_j}^1, v_{w_j}^2, ..., v_{w_j}^h\}$, $(i = 1, ..., k)$ represents a set of context words in the i-th cluster of the word w_j.

Definition 1 *w_i and w_j $(i \neq j)$ are called synonyms if:*

- *$w_i \in t_x$ and $w_j \in t_y$, $(x \neq y)$*
  And
- *$\mathcal{G}_{w_i} = \mathcal{G}_{w_j}$*

Definition 2 *The word embedding of synonym word s, denoted by z_s, is a transform of synonym s into a d-dimensional vector by calculating the average vector of context words of this synonym. The synonym word embedding z_s is defined as follows:*

$$z_s = \mathcal{AVG}(vector(\mathcal{M}, \mathcal{G}_s)) \tag{6}$$

where $\mathcal{AVG}$ is a function to calculate the average vector, vector is a mapping function that is used to map each context word into one word embedding.

3.3 Research Question

In this study, we attempt to answer the main question: *How to determine the vector representations of synonym words?*. This question is divided into the following two questions:

1. *How to identify the synonym words by using their context words?*
2. *How to convert the synonym words into numerical vectors?*

4 Proposed Method

In this section, we present a methodology to enrich word embeddings by adding synonym information. The workflow of our method is illustrated in Fig.1.

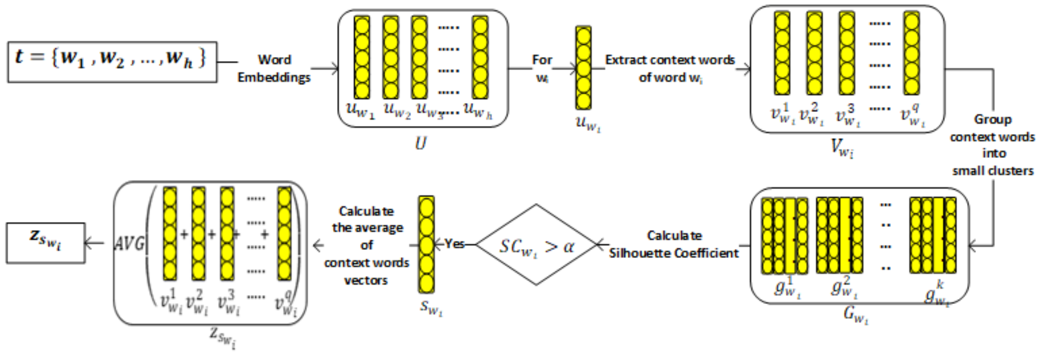
$t = \{w_1, w_2, \dots, w_h\}$
Word Embeddings
u_{w_1} u_{w_2} u_{w_3} ... u_{w_h}
U
For w_i
u_{w_i}
Extract context words of word w_i
$v^1_{w_i}$ $v^2_{w_i}$ $v^3_{w_i}$ ..... $v^q_{w_i}$
V_{w_i}
Group context words into small clusters
$g^1_{w_i}$ $g^2_{w_i}$ $g^k_{w_i}$
G_{w_i}
Calculate Silhouette Coefficient
$SC_{w_i} > \alpha$
Yes
s_{w_i}
Calculate the average of context words vectors
AVG
$z_{s_{w_i}}$

Fig. 1. Workflow of the proposed method.

Our proposed method consists of three main steps. First, the context words of the synonym candidates are extracted using a context window to scan the entire corpus. Second, these context words are grouped into small clusters using the LDA method. Finally, synonyms are extracted and converted into vectors from the synonym candidates based on their context words.

4.1 Word Embeddings

Word embeddings are created using the available text representation models that are used to convert words in a corpus into a vector space. Here, we used the model introduced in [20] to obtain word embeddings for our corpus. The detailed steps to create these vectors are presented in Fig.2. Its components are described as follows.

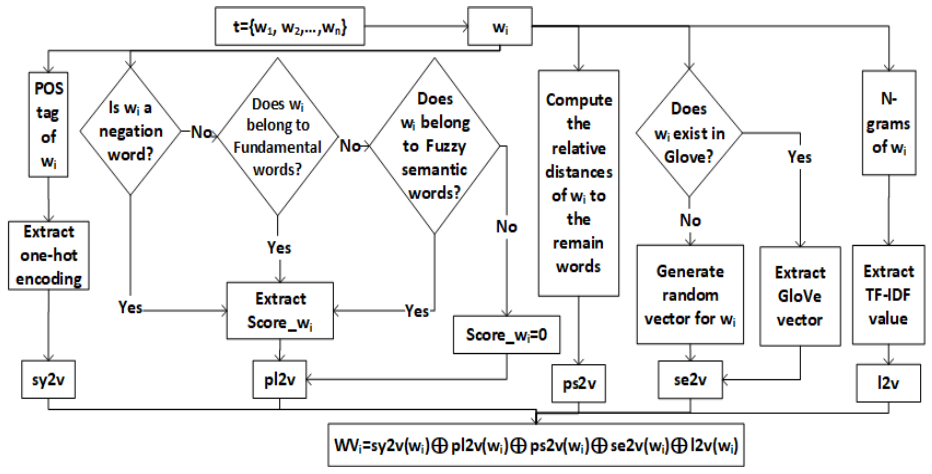

Fig. 2. The word embeddings model architecture.

Parameter *l2v* denotes the lexicon vector. To create the lexicon vector of a word, first, n-grams starting from this word, such as 1-gram, 2-grams, and 3-grams, are extracted. Then, the term frequency-inverse document frequency (TF-IDF) value of these n-grams is calculated. Finally, the TF-IDF values of n-grams are concatenated into one vector.

Parameter *sy2v* denotes the word-type vector of a word. This vector is built as follows: First, the POS tag of this word is identified. Then, this word is converted into a one-hot encoding vector based on the position of the corresponding POS tag.

Parameter *se2v* denotes the semantic vector. This vector is created based on the GloVe embeddings [17]. If this word exists in the GloVe dataset, the word vector of this word is extracted and assigned to the semantic vector. If not, a random vector of this word is created and assigned to the semantic vector.

Parameter *pl2v* denotes the polarity sentiment vector of a word. To create this vector, first, the kind of word of this word is determined. Then, the sentiment score of this word is calculated. Finally, the polarity sentiment vector is created based on this sentiment score.

Parameter *ps2v* denotes the position vector. To build the position vector of the word, first, the position of this word is extracted by calculating the distances from this word to the remaining words in a tweet. Then, the position vector is created based on these distances.

4.2 Context Words Extraction

For each w : let $region(w, d)$ represent a context region of word w with the region length as $2 \times r + 1$, where r denotes the window size for a region. In this study, synonym words are identified via the position of their context words. The aim of this phase is to find set $\mathcal{V} = \{\mathcal{V}_{w_1}, \mathcal{V}_{w_2}, ..., \mathcal{V}_{w_m}\}$. Therefore, any word in a region of text can become the context word of a target word. The context words set is determined according to the following equation.

$$\mathcal{V}_{w_i} = \bigcup_{r=1}^{2\times r+1} region(w_i, r) \tag{7}$$

Hence,

$$\mathcal{V} = \bigcup_{i=1}^{h} \mathcal{V}_{w_i} \tag{8}$$

The positions of context words of word w are determined determined according to Algorithm 1.

Algorithm 1 Context words extraction

```
Input: $\mathcal{M}$;
Output: $\mathcal{P}$;
 1: for $i = 1$ to $m$ do
 2:     for each $w_i$ do
 3:         $w_c := w_i$;
 4:         for $\theta = 1$ to $2 \times r + 1$ do
 5:             if $w_c \in region(w_i, \theta)$; then insert $w_c$ into $\mathcal{V}_{w_i}$;
 6:         end for;
 7:     end for;
 8:     insert $\mathcal{V}_{w_i}$ into $\mathcal{V}$;
 9: end for;
10: for $i = 1$ to $m$ do
11:     for $\mathcal{V}_{w_i} \in \mathcal{V}$ do
12:         $\mathcal{P} = \{[w_i, \mathcal{V}_{w_i}]\}$
13:     end for;
14: end for;
15: return $\mathcal{P} = \{[w_1, \mathcal{V}_{w_1}], [w_2, \mathcal{V}_{w_2}], ..., [w_m, \mathcal{V}_{w_m}]\}$
```

4.3 Clustering context words

Given a set of context vocabularies denoted by $\mathcal{V} = \{\mathcal{V}_{w_1}, \mathcal{V}_{w_2}, ..., \mathcal{V}_{w_m}\}$, assuming that we have a target word w and its context vocabulary $\mathcal{V}_w = \{v_w^1, v_w^2, ..., v_w^q\}$ in the mapping table $\mathcal{P}$. In this phase, we use the LDA model [2] to group the context words of the target word w into small clusters, denoted by $\mathcal{G}_w = \{g_w^1, g_w^2, ..., g_w^k\}$, where $g_w^i = \{v_w^1, v_w^2, ..., v_w^h\}$, ($h < q$ and $i = 1, .., k$) represent a set of words that are grouped into the i-th class of the context vocabulary of the word w. The

LDA model can be generated using the following process [8, 19]: Let m denote the size of the vocabulary and n the total number of context vocabularies in $\mathcal{V}$. A statistical topic model represents the words in a collection of tweets as mixtures of k topics, words within context vocabularies $w_{e,q}, (e = 1, ..., m; q = 1, ..., n)$ are observed variables while the probabilistic distribution over words of each latent topic $\varphi_\ell(\ell = 1, ..., k)$ with hyper parameter γ, the topic distribution per tweet $\theta_e, (e = 1, ..., m)$ with hyperparameter δ and the perword topic assignment $z_{e,q}$ are hidden variables. For each tweet, the words are created by the following steps: First, a distribution over topics is randomly selected. A topic is randomly selected for each word in the tweet based on the distribution over topics. Second, the hidden random variables (φ_ℓ and θ_e) are not observed that could be learned through Gibbs sampling method[7] via maximizing the probability $p(\mathcal{V}|\delta, \gamma)$ as the following equation:

$$p(\mathcal{V}|\delta, \gamma) = \prod_{e=1}^{m} \int p(\theta_e|\delta)(\prod_{q=1}^{n} \sum_{z_{e,q}} p(z_{e,q}|\theta_e)p(w_{e,q}|z_{e,q}, \gamma))d\theta_e \tag{9}$$

The LDA model provides the outputs including k sub-clusters of context words that belong to the given cluster $\mathcal{G}_w = \{g_w^1, g_w^2, ..., g_w^k\}$, the word distribution per topic $\varphi_\ell, (\ell = 1, ..., k)$ and the topic distribution per the context vocabulary $\theta_e, (e = 1, ..., m)$. The steps to cluster the context words are presented as the following Algorithm 2.

Algorithm 2 Clustering context words

```
Input: $\mathcal{P}$;
Output: $\mathcal{G}_{w_e}$, $\varphi_{w_e}$, $\theta_{w_e}$;
1: for $\ell = 1$ to $k$ do
2:   for $e = 1$ to $m$ do
3:     $g_{w_e}^\ell$, $\varphi_\ell$, $\theta_e$ = LDA($\mathcal{P}$);
4:     insert $g_{w_e}^\ell$ into $\mathcal{G}_{w_e}$;
5:   end for;
6:   assign $\varphi_\ell$ to $\varphi_{w_e}$;
7:   assign $\theta_e$ to $\theta_{w_e}$;
8: end for;
9: return $\mathcal{G}_{w_e}$, $\varphi_{w_e}$, $\theta_{w_e}$
```

4.4 Synonym Words Extraction

In this study, the synonym words are extracted by calculating the Silhouette Coefficient [23]. Thus, for a cluster of context words of a target word $g_{w_e}^\ell, (e = 1, ..., m; \ell = 1, ..., k)$, we have to calculate the Silhouette coefficient of each $w_e \in g_{w_e}^\ell$. Let

[7] https://gist.github.com/mblondel/542786#file-lda_gibbs-py

$\mathcal{SC}_{w_e}$ denote the Silhouette coefficient of word w_e in cluster $g^{\ell}_{w_e}$.

$$\mathcal{SC}_{w_e} = \frac{1}{m}\sum_{e=1}^{m}(\mathcal{S}w_{g^{\ell}_{w_e}}) \tag{10}$$

where
$$\mathcal{S}w_{g^{\ell}_{w_e}} = \frac{b_{w_e} - a_{w_e}}{max(a_{w_e}, b_{w_e})} \tag{11}$$

where
$$a_{w_e} = \frac{1}{n_\ell - 1}\sum_{w_f \in g^{\ell}_{w_e} (f \neq e)} \mathcal{D}_{ef} \tag{12}$$

$$b_{w_e} = min_{h \neq \ell}(\frac{1}{n_h}\sum_{w_f \in g^{h}_{w_e}} \mathcal{D}_{ef}) \tag{13}$$

where
$$\mathcal{D}_{ef} = \sqrt{\sum_{e,f=1}^{m}(Q_{w_e} - Q_{w_f})^2} \tag{14}$$

$$Q_{w_e} = \varphi_{w_e} \times \theta_{w_e}; Q_{w_f} = \varphi_{w_f} \times \theta_{w_f} \tag{15}$$

where $\mathcal{D}_{ij}$ is the Euclidean distances for all pairs of the words in cluster $g^{\ell}_{w_e}$; $n_\ell(n_h)$ is the number of words in the ℓ-th (h-th) cluster. The value of the Silhouette Coefficient is in the interval [-1,1]. A higher value implies a better assignment of words into clusters. Therefore, in this study, a word is decided as a synonym word when the value of the Silhouette coefficient is equal to 1. The steps to determine the synonym words are illustrated in Algorithm 3:

Algorithm 3 Synonym words extraction

```
Input: $\mathcal{G}_{w_e}, \varphi_{w_e}, \theta_{w_e}$;
Output: $\mathcal{S}_{w_e}$;
 1: for $\ell = 1$ to $k$ do
 2:     for $e = 1$ to $m$ do
 3:         $\mathcal{SC}_{w_e} = Silhouette\_Coefficient(g^{\ell}_{w_e})$;
 4:     end for;
 5:     if $\mathcal{SC}_{w_e} > \alpha$ then
 6:         $w_e$ is determined as a synonym word;
 7:         $s_{w_e} := w_e$;
 8:     end if
 9:     insert $s_{w_e}$ into $\mathcal{S}_{w_e}$;
10: end for;
11: return $\mathcal{S}_{w_e}$;
```

4.5 Synonym Words Representation

Using the aforementioned steps, the synonym words are extracted from the tweets. Next, we have to determine a way to convert these synonym words into numerical vectors. In this study, the synonym words are represented as vectors by calculating their context words' average vectors. The equation to calculate the context words' average vectors is described in Definition 2. The overall algorithm of the synonym word representation is presented in the following Algorithm 4.

Algorithm 4 Synonym words representation

```
Input: a set of synonym words $\mathcal{S}_{w_e}$;
       a set of context words of synonym word $s_{w_e}$, denoted by $\mathcal{G}_{s_{w_e}}$, where $s_{w_e} \in \mathcal{S}_{w_e}$;
Output: $z_{s_{w_e}}$;
1: for $e = 1$ to $m$ do
2:     $z_{s_{w_e}} = \mathcal{AVG}(vector(\mathcal{M}, \mathcal{G}_{s_{w_e}}))$;
3: end for;
4: return $z_{s_{w_e}}$;
```

5 Experiment

5.1 Data Acquisition

The proposed method was applied to tweet data. The tweets in Semeval-2013[8] were used to train our proposal. Then, the unnecessary factors in tweets, such as punctuation, retweet marks, URLs, hashtags, and query terms were discarded. The Python emoji package[9] was used to replace each emoji with descriptive text. Tweets often include acronyms, spelling errors, and symbols. It is necessary to correct them. We fixed these spellings using the Python-based Aspell library[10]. In addition, to evaluate the performance of our method, we experimented with three English word datasets, namely, WordSim-353 [5], RG-65 [24], and SimLex-999 [10]. These datasets were obtained from Svoboda *et al.* [27][11]. WordSim-353 includes 353 word pairs, including both concepts and named entities. RG-65 includes 65 word pair similarities. The SimLex-999 dataset is composed of 999 word pairs, 666 of which are noun pairs.

5.2 Evaluation Method

We evaluated the performance of our proposal for the task of word similarity (relatedness). This evaluation method was implemented by computing Spearman's rank correlation coefficient [25] between annotators and the obtained vectors of our system. Furthermore, to prove the quality of our approach, we compared our synonym word embedding model with corresponding state-of-the-art models by implementing the following baseline methods: Baseline 1: A Word2Vec model that is trained on the entire corpus without considering the synonym information. Baseline 2: A GloVe model that is also trained on the entire corpus without considering the synonym information. Baseline 3: A text representation model regarding tweets containing fuzzy sentiment that considers elements such as lexicon, word-type, semantic, position, and sentiment polarity of words [20].

[8] https://www.kaggle.com/azzouza2018/semevaldatadets?select=semeval-2013-train.csv

[9] https://pypi.org/project/emoji/

[10] https://pypi.org/project/aspell-python-py2/

[11] https://github.com/Svobikl/global_context/tree/master/AnalogyTester/evaluation_data

5.3 Training Setup

In this study, the tweets in the dataset were tokenized into separate words using the NLTK package[12]. This model was trained on 10 iterations. After training, the vector dimension for our model was set to $d = 300$. In addition, we used a context window of size 5 to the left and 5 to the right from the target word to extract the context words. Additionally, the threshold α is chosen by 0.7. All above parameters were set manually following the experiments. We conducted an exhaustive search for d from 50 to 400, α from 0.5 to 1. In each trial, we adjusted the thresholds with increments of 50 and 0.1 for d and α, respectively. An evaluation measure was necessary to select the highly reliable instances of d and α in an exhaustive search. Therefore, in this study, the basis of choosing the above parameters' value is based on the value of the word-similar score. The highest word-similar score was obtained for the threshold $d = 300$ and $\alpha = 0.7$. Selecting a higher or lower values for d and α would result in a misprediction of more synonyms. The baseline methods were also trained with the same dataset.

5.4 Result and Discussion

The results for our method and the baseline methods are presented in Table 1. Notably, some words in testing datasets were not included in our training data. Therefore, we could not obtain the vector representation of these words. Hence, for these words, we created random vectors to provide comparable results.

Table 1. Word similarity results (%).

Method	WordSim-353	RG-65	SimLex-999
Baseline 1	76.12	69.31	42.26
Baseline 2	73.64	65.12	41.25
Baseline 3	78.73	69.32	40.57
Our approach	79.54	72.32	41.20

According to Table 1, it can be seen that for the WordSim-353 dataset, our approach can improve the performance of the Baselines 1,2, and 3 by 3.42%, 5.9%, and 0.81%, respectively. Besides, for the RG-65 dataset, our method can increase the word similarity accuracy of the baseline methods by up to 7.2% (for Baseline 2), by at least 3% (for Baseline 3). However, for the SimLex-999 dataset, although the performance of our proposal was higher than Baseline 3, but it was lower than the remaining methods by up to 1.06% comparison with Baseline 1, by at least 0.05% comparison with Baseline 2.

As our assessment, our approach outperformed the baselines on the RG-65 and WordSim-353 datasets, but not on the SimLex-999 dataset. In this dataset, the performance of our method was lower than that of Baseline 1 and Baseline 2 methods. This regard was because words in the SimLex-999 dataset were common words for good vectors obtained without exploiting synonym information.

[12] https://www.nltk.org/_modules/nltk/tokenize/api.html

When evaluating less frequent words, we noted that using the context words of a target word helped to learn good word vectors.

In general, our method proved the role of synonym information when enriching word representations. Our approach improved the performance of the prior techniques, but not always, owing to the imbalance of synonym frequency in the datasets.

6 Conclusion and Future Work

We improved the vector representation of words by adding the synonym information. This improvement focuses on the extraction and presentation of synonyms based on their context words. We show that our method has extracted synonym words based on grouping their context words. The synonym words have been represented to vectors by computing the average vector from vectors of their context words. By comparing to recent word representation methods, we proved that our proposal achieved a quite good performance on a given task in terms of word similarity. The main limitation is that we have not compared this proposal's performance to other methods of synonym representations because of the difficulty in determining similar methods for comparison. We will open-source our implementation to make easy the comparison of future work on learning synonym vectors.

References

1. Al-Twairesh, N., Al-Negheimish, H.: Surface and deep features ensemble for sentiment analysis of arabic tweets. IEEE Access **7**, 84122–84131 (2019)
2. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of machine Learning research **3**(Jan), 993–1022 (2003)
3. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5**, 135–146 (2017)
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
5. Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., Ruppin, E.: Placing search in context: The concept revisited. In: Proceedings of the 10th international conference on World Wide Web. pp. 406–414 (2001)
6. Gong, H., Bhat, S., Viswanath, P.: Enriching word embeddings with temporal and spatial information. arXiv preprint arXiv:2010.00761 (2020)
7. Guo, S., Yao, N.: Polyseme-aware vector representation for text classification. IEEE Access **8**, 135686–135699 (2020)
8. Hamzehei, A., Wong, R.K., Koutra, D., Chen, F.: Collaborative topic regression for predicting topic-based social influence. Machine Learning **108**(10), 1831–1850 (2019)
9. Harris, Z.S.: Distributional structure. Word **10**(2-3), 146–162 (1954)
10. Hill, F., Reichart, R., Korhonen, A.: Simlex-999: Evaluating semantic models with (genuine) similarity estimation. Computational Linguistics **41**(4), 665–695 (2015)

11. Jianqiang, Z., Xiaolin, G., Xuejun, Z.: Deep convolution neural networks for twitter sentiment analysis. IEEE Access **6**, 23253–23260 (2018)
12. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., Mikolov, T.: Fasttext. zip: Compressing text classification models. arXiv preprint arXiv:1612.03651 (2016)
13. Kim, Y.: Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014)
14. Kundi, F.M., Ahmad, S., Khan, A., Asghar, M.Z.: Detection and scoring of internet slangs for sentiment analysis using sentiwordnet. Life Science Journal **11**(9), 66–72 (2014)
15. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360 (2016)
16. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
17. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
18. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. arXiv preprint arXiv:1802.05365 (2018)
19. Phan, H.T., Nguyen, N.T., Tran, V.C., Hwang, D.: An approach for a decision-making support system based on measuring the user satisfaction level on twitter. Information Sciences (2021). https://doi.org/https://doi.org/10.1016/j.ins.2021.01.008
20. Phan, H.T., Tran, V.C., Nguyen, N.T., Hwang, D.: Improving the performance of sentiment analysis of tweets containing fuzzy sentiment using the feature ensemble model. IEEE Access **8**, 14630–14641 (2020)
21. Qi, Y., Sachan, D.S., Felix, M., Padmanabhan, S.J., Neubig, G.: When and why are pre-trained word embeddings useful for neural machine translation? arXiv preprint arXiv:1804.06323 (2018)
22. Rezaeinia, S.M., Rahmani, R., Ghodsi, A., Veisi, H.: Sentiment analysis based on improved pre-trained word embeddings. Expert Systems with Applications **117**, 139–147 (2019)
23. Řezanková, H.: Different approaches to the silhouette coefficient calculation in cluster evaluation. In: 21st International Scientific Conference AMSE Applications of Mathematics and Statistics in Economics 2018. pp. 1–10 (2018)
24. Rubenstein, H., Goodenough, J.B.: Contextual correlates of synonymy. Communications of the ACM **8**(10), 627–633 (1965)
25. Sedgwick, P.: Spearman's rank correlation coefficient. Bmj **349**, g7327 (2014)
26. Svoboda, L., Brychcın, T.: Improving word meaning representations using wikipedia categories. Neural Network World **523**, 534 (2018)
27. Svoboda, L., Brychcín, T.: Enriching word embeddings with global information and testing on highly inflected language. Computación y Sistemas **23**(3) (2019)
28. Ulčar, M., Robnik-Šikonja, M.: High quality elmo embeddings for seven less-resourced languages. arXiv preprint arXiv:1911.10049 (2019)
29. Wang, B., Wang, A., Chen, F., Wang, Y., Kuo, C.C.J.: Evaluating word embedding models: methods and experimental results. APSIPA transactions on signal and information processing **8** (2019)