

# Beyond word2vec: Distance-graph Tensor Factorization for Word and Document Embeddings

Suhang Wang  
 Pennsylvania State University  
 University Park, PA  
 szw494@psu.edu

Charu Aggarwal  
 IBM T.J. Watson Research Center  
 Yorktown Heights, NY  
 charu@us.ibm.com

Huan Liu  
 Arizona State University  
 Tempe, AZ  
 huan.liu@asu.edu

## ABSTRACT

The *word2vec* methodology such as Skip-gram and CBOw has seen significant interest in recent years because of its ability to model semantic notions of word similarity and distances in sentences. A related methodology, referred to as *doc2vec* is also able to embed sentences and paragraphs. These methodologies, however, lead to different embeddings that cannot be related to one another. In this paper, we present a tensor factorization methodology, which simultaneously embeds words and sentences into latent representations in one shot. Furthermore, these latent representations are concretely related to one another via tensor factorization. Whereas *word2vec* and *doc2vec* are dependent on the use of contextual windows in order to create the projections, our approach treats each document as a structural graph on words. Therefore, all the documents in the corpus are jointly factorized in order to simultaneously create an embedding for the individual documents and the words. Since the graphical representation of a document is much richer than a contextual window, the approach is capable of designing more powerful representations than those using the *word2vec* family of methods. We use a carefully designed negative sampling methodology to provide an efficient implementation of the approach. We relate the approach to factorization machines, which provides an efficient alternative for its implementation. We present experimental results illustrating the effectiveness of the approach for document classification, information retrieval and visualization.

## CCS CONCEPTS

• Information systems → Data mining;

## KEYWORDS

Word Embedding; Document Embedding; Pairwise Factorization

## ACM Reference Format:

Suhang Wang, Charu Aggarwal, and Huan Liu. 2019. Beyond word2vec: Distance-graph Tensor Factorization for Word and Document Embeddings. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3357384.3358051>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

CIKM '19, November 3–7, 2019, Beijing, China  
 © 2019 Association for Computing Machinery.  
 ACM ISBN 978-1-4503-6976-3/19/11...\$15.00  
<https://doi.org/10.1145/3357384.3358051>

## 1 INTRODUCTION

In recent years, the problem of finding word and document embeddings has gained increasing attention in the text domain. While matrix factorization is a traditional methodology that is used for word embeddings, methods like SVD [12], LSA [15], and NMF are designed to work with the bag-of-words representation. A detailed discussion of these different methods may be found in [1].

For more semantically-oriented applications, sequence-centric embeddings have been proposed in the literature. The traditional approach of sequential embeddings is to use substring kernels in order to embed the documents in multidimensional space [20]. Such embedding methods are typically designed at the document level rather than at the word level. At the word level, neural embedding methods such as *word2vec* (Skip-gram and CBOw) [24] are used in which the words are related to contextual windows on either side of the word. The size of this context window is typically of the order of about 10 words, and it decides the level of sequential relationships that captured with such an approach. Recently, this broader approach has been extended to document-level embeddings with the *doc2vec* framework [16]. The use of matrix factorization has also been recently proposed for embedding words [17, 27] with the use of context windows. Interestingly, it has been shown that some variants of the *word2vec* framework are really matrix factorization methods under the covers [11, 17, 25]. In particular, the mutual information between words and contexts is determined, and a shifted positive version (SPPMI) matrix is constructed between words and context pairs [17]. The factorization of this matrix can be shown to be equivalent to *word2vec*. Another example of a method that uses matrix factorization is GloVe [27], which factorizes a count-based matrix on context windows in order to create the embedded representation. GloVe and *word2vec* both provide competitive performance. Such embeddings can often capture many semantic characteristics of words that cannot be achieved by traditional matrix factorization of document-word matrices. For example, one can add and subtract semantic concepts using such an embedding  $F(\cdot)$  as  $F(\text{king}) - F(\text{queen}) = F(\text{man}) - F(\text{woman})$ . One is able to learn embeddings satisfying these properties because sequences encode a lot more contextual information than is available in the bag-of-words representation. A separate method, referred to as *doc2vec* [16], is designed for embedding at the document level rather than at the word level with the similar distributional representation idea as *word2vec*.

In spite of the recent popularity of the *word2vec* and *doc2vec* methods, they have several weaknesses:

- (1) The use of context windows can sometimes be a rather blunt approach to capturing the sequential relationships between words. Part of the problem is that the words within a context window are treated in a uniform way, and the relative distance of a word from the target word is not used effectively.

- (2) The *word2vec* framework treats words following or preceding a target word in symmetric fashion. In practice, it is expected that there would be significant syntactic differences between the words preceding or following a target word.
- (3) Methods like *word2vec* create word embeddings, whereas methods like *doc2vec* create document embeddings. Using *both* word ordering and document membership in a single framework is inherently a more sophisticated model than separately creating word embeddings and document embeddings with *aggregated* count statistics (as in the currently available models). The approach discussed in this paper uses a tensor to represent the sequence-centric and document-membership within a single representation, which is then factorized. A tensor is far more powerful than a count-based matrix (used by methods like GloVe) in capturing nuances of the data.

In this paper, we will explore the use of a distance graph and negative sampling in order to capture these relationships in detail. One advantage of the distance graph approach is that it does not treat all words within a context window in a uniform way, which allows it to use the relative distances between words in a more refined fashion. Furthermore, the use of tensors provides a more powerful framework. In spite of the use of tensors, we provide an efficient implementation with negative sampling in which each cycle of the gradient descent requires time that is linearly proportional to the size of the corpus. Furthermore, the approach can be related to factorization machines, which provides an alternative methodology for implementation. The main contributions are:

- We provide a principled approach for joint word and document embedding by exploring the use of distance graph, which can capture word relations in a more refined fashion;
- We propose a novel framework *Distance Graph Tensor Factorization* (DGTF), which utilizes efficient pairwise shared matrix factorization on distance graphs to simultaneously learn word and document embedding and the approach can be related to factorization machines; and
- We conduct extensive experiments on real-world datasets to demonstrate the effectiveness of DGTF for various tasks <sup>1</sup>.

This paper is organized as follows. Section 2 introduces background and the distance graph. Section 3 gives the details of DGTF. Section 4 conducts experiments to evaluate the effectiveness of DGTF. Section 6 concludes the paper with future work.

## 2 THE DISTANCE GRAPH

The distance graph [2] is a method for representing the sequential relationships between words. The distance graph can be created either at the document, paragraph, or sentence level. Furthermore, all distance graphs are defined over the same subset of nodes, which are defined by the lexicon. In other words, each word corresponds to one node in the lexicon, and the entire set of nodes has cardinality equal to the size  $d$  of lexicon. For now, let us consider a set of documents  $D_1, D_2, \dots, D_n$ , each of which needs to be embedded using the same lexicon. A distance graph (specific to a particular document  $D_r$ ) is a directed graph in which an edge occurs from node  $i$  to  $j$ , if and only if the word  $j$  occurs after word  $i$  within a specific threshold distance  $k \geq 1$  in the document  $D_r$ . Here, the threshold  $k$  defines the order of the distance graph. Using  $k = 1$  corresponds to

using only bigrams as the edges in a sentence. Larger values of  $k$  allow the use of skip grams. Furthermore, the weight of the edge in the distance graph decays with the number of skips. Specifically for any word from node  $i$  to  $j$  with positions  $p_i$  and  $p_j$  in the document ( $p(j) > p(i)$ ), the weight of the corresponding edge is given by  $\lambda^{p(j)-p(i)}$ . Here,  $\lambda < 1$  is a decay parameter that controls the effect of distances between words on their embedded representations. Note that if the same pair of words occur multiple times in a document with a distance of at most  $k$  between them, then the edge weight of the words increases. Specifically, if  $m_1, m_2, \dots, m_t$  are the distances between a particular word pair  $(i, j)$  using  $t$  different occurrences in the  $r$ -th distance graph, then the weight  $a_{ij}^r$  between the word pair is defined as follows:

$$a_{ij}^r = \sum_{s=1}^t \lambda^{m_s} \quad (1)$$

It is also noteworthy that before creating the distance graph, very frequent words and stop words are dropped from the documents. This is done in order to ensure that the statistics of such words do not dominate the final embedding.

An example of a distance graph created from the nursery rhyme “*Mary had a little lamb...*” is shown in Figure 1. Distance graphs of different orders are also given in the figure. It is noteworthy that such embeddings at the sentence or document level capture a lot more information than is available using contextual window-methods like *word2vec* because (i) for *word2vec*, the words within a context window are treated in a uniform way while graph distances can capture the relative distance of words in the context window by assigning different weights to words of different distance; and (ii) *word2vec* treats words following or preceding a target word in symmetric fashion, which ignores the significant syntactic differences between words following or preceding a target word; while distance graph exploits an asymmetric way to capture this information.

For some applications, it might be desirable to work with sentence-level distance graphs, whereas for other applications, it might be desirable to work with document-level graphs. In either case, the embedding is created at either the sentence or the document level. However, the distance graph is never created at the corpus level. It is noteworthy that most word embedding techniques create statistics at the corpus level in order to extract the representations [17, 27, 34]. Although this simplification has the advantage of being able to work with matrices (instead of tensors), we argue that it also loses a lot of relevant information in the corpus.

The specific level of granularity (e.g., sentence, paragraph, or document) is orthogonal to the details of the methodology discussed in this paper. Therefore, we will generally refer to the  $n$  different units of text as “text segments”, and it is assumed that distance graphs have been created for these  $n$  text segments. The adjacency matrices for these distance graphs are defined as  $A_1 \dots A_n$ . The matrix  $A_i$  is a  $d \times d$  matrix for the  $i$ -th text segment in which the weight of a particular entry is its corresponding weight in the distance graph of that segment (see Equation 1). The construction of the distance graphs is a preprocessing step towards creating the embeddings. Therefore, throughout the remaining sections, we will assume that the distance graphs of all the text segments are already available for creating the embeddings.

Each matrix  $A_i$  is *extremely* sparse, particularly for short text segments. For example, a distance graph of order 3 for a text segment

<sup>1</sup>The code of DGTF will be released upon the acceptance of the paper in the first authors homepage.

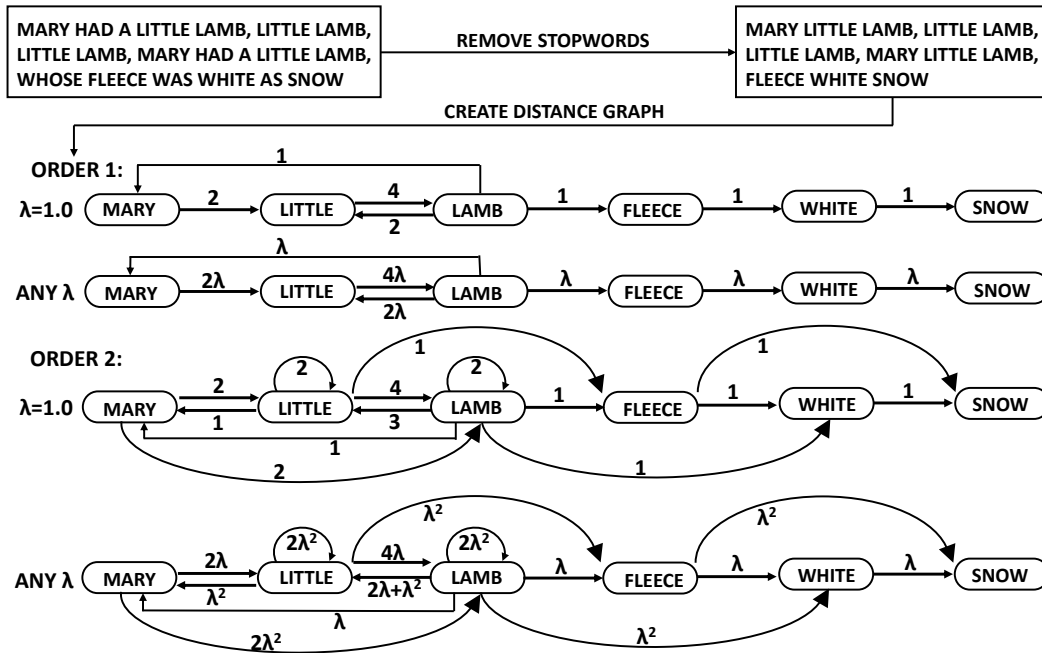


Figure 1: Distance graphs of orders  $k = \{1, 2\}$ .

of 30 words will contain at most 84 edges, whereas the dimensionality  $d$  of each side of the matrix may be of the order of  $10^4$ .  $d$  is the vocabulary size of the number of lexicons. In fact, most nodes (words) will not have a single incident edge because they are not included in the text segment. As we will see later, it is important to make use of this sparsity during the process of learning the embeddings. Specifically, we will use the matrices  $A_1, \dots, A_n$  to create an  $d \times d \times n$  tensor, which will be factorized in order to create the document and word embeddings simultaneously. The use of a tensor is crucial in creating a representation that can simultaneously discover word and document embeddings.

### 3 TENSOR FACTORIZATION FOR DOCUMENT AND WORD EMBEDDINGS

For the purpose of this section, we will assume that the data has already been preprocessed to create the distance graphs. The preprocessing can be done at the document, paragraph, or sentence level, and in each case a set of distance graphs is created. Assume that there are a total of  $n$  segments, which create the distance graphs with adjacency matrices denoted by  $A_1 \dots A_n$ . Each  $A_i$  is assumed to be a  $d \times d$  adjacency matrix, which is extraordinarily sparse.

How can one simultaneously discover both word and document embeddings? Most of the known methods [17, 25, 27] for matrix factorization are not rich enough to incorporate sequence information for *simultaneously* creating word and document embeddings. In order to capture this level of complexity, it becomes essential to use *tensor* factorization as opposed to straightforward matrix factorization. The basic idea is to represent the matrices  $A_1 \dots A_n$  as the  $d \times d$  slices of a tensor of size  $d \times d \times n$ . Let the  $(i, j)$ -th entry of  $A_r$  be denoted by  $a_{ij}^r$ . The entire  $d \times d \times n$  tensor is denoted by  $\mathcal{A}$ . The  $(i, j, r)$ -th cell in  $\mathcal{A}$  corresponds to the edge  $(i, j)$  in the  $r$ -th

distance graph. It is noteworthy that this tensor is *extremely* sparse, because the vast majority of the cells are 0s.

Although generic tensor factorization can be expensive [14], the nice part of the tensor factorization of an order-3 tensor is that one can simulate it with *pairwise shared matrix factorization* [30]. Later, we will show the relationship of this problem with factorization machines, and show how one can use off-the-shelf factorization machines for an effective embedding. Therefore, we will first introduce the types of matrices that we will use for the factorization process. In order to perform the factorization, we have three types of factor matrices of rank  $p$ :

- The matrix  $U$  is a  $d \times p$  matrix, which contains the *outgoing* node factors of all the nodes. Intuitively, each row of this factor matrix captures the latent factors of the outgoing edges of nodes of all distance graphs in the collection. The  $(i, s)$ -th entry of  $U$  is denoted by  $u_{is}$ , and the  $i$ -th row of  $U$  is denoted by  $\bar{u}_i = [u_{i1}, u_{i2}, \dots, u_{is}]$ . We can also view  $\bar{u}_i$  as a portion of the embedding of word  $i$  that captures the information in the context *following* word  $i$ .
- The matrix  $V$  is a  $d \times p$  matrix, which contains the *incoming* node factors of all the nodes. Intuitively, each row of this factor matrix captures the latent factors of the incoming edges of a node in the distance graph. The  $(j, s)$ -th entry of  $V$  is denoted by  $v_{js}$ , and the  $j$ -th row of  $V$  is denoted by  $\bar{v}_j = [v_{j1}, v_{j2}, \dots, v_{js}]$ . Like the matrix  $U$ , the rows of  $V$  also provide the embeddings of different words but they capture the context *preceding* the words.
- The matrix  $W$  is an  $n \times p$  matrix that captures the embeddings of the  $n$  documents (or units of text being considered in the specific application at hand). The  $(i, j)$ -th entry of  $W$  is denoted by  $w_{ij}$ . The  $r$ -th row  $\bar{w}_r = [w_{r1}, w_{r2}, \dots, w_{rp}]$  of  $W$  contains a  $p$ -dimensional embedding of the  $r$ -th document in the corpus.

Like *word2vec*, the embeddings in  $U$  and  $V$  reflect the sequence characteristics of the entire collection. Such word embeddings are able to capture many semantic characteristics of the corpus.

This is a third order tensor, which can be factorized using the canonical polyadic decomposition (CPD) or the Tucker decomposition [14]. However, these decomposition methods are expensive. Fortunately, a third order tensor factorization can be simulated with additive matrix factorization because the second-order factors are 2-dimensional matrices [30]. Specifically, we can factorize the  $d \times d \times n$  tensor with entries  $a_{ij}^r$  by predicting the entries as:

$$\begin{aligned} \hat{a}_{ij}^r &\approx (UV^T)_{ij} + (UW^T)_{ir} + (VW^T)_{jr} \\ &= \sum_{s=1}^p [u_{is}v_{js} + u_{is}w_{rs} + v_{js}w_{rs}] \end{aligned}$$

Note the circumflex “ $\hat{\cdot}$ ” on top of  $\hat{a}_{ij}^r$  is to indicate that it is predicted value. The objective function for optimization is set up by summing up the squares of the errors in the aforementioned relationship, and then learning the entries of  $U$ ,  $V$ , and  $W$  in order to minimize this error. Therefore, the objective function  $J$  (for minimization) can be written as follows:

$$J = \frac{1}{2} \sum_r \sum_i \sum_j (a_{ij}^r - \sum_{s=1}^p [u_{is}v_{js} + u_{is}w_{rs} + v_{js}w_{rs}])^2 \quad (2)$$

This objective function needs to be minimized by using gradient descent. The main problem with this objective function is that it is computationally too expensive to evaluate, as it has too many terms. For example, for a relatively modest text corpus with  $d = 10^5$ ,  $n = 10^5$ , and  $p = 100$ , the objective function has  $10^{17}$  terms. State-of-the-art hardware, which runs at  $10^{10}$  cycles per second, would require substantially more than  $10^7$  seconds ( $\approx 31$  days) simply to read all the entries of the tensor  $\mathcal{A}$  at least once, let alone compute the errors and optimize them.

Fortunately, most of the cell values  $a_{ij}^r$  are 0s because of the sparsity of the distance graph. The number of nonzero cells in  $\mathcal{A}$  is, however, quite modest. Zero entries are not irrelevant because they contribute to the errors in the cells of the distance graph. For example, it is possible to create trivial (i.e., overfitting) solutions for  $U$ ,  $V$ , and  $W$ , in some special cases in which all nonzero entries in the distance graphs are 1s or are very close values. Therefore, we need a method for *negative sampling* of the zero values of the cells in  $a_{ij}^r$ . We can view this type of negative sampling method as the distance graph analog of what is done in *word2vec* and noise contrastive estimation [25]. How can we sample the negative cells in a way that is meaningful for the application at hand? One solution is to sample the cells randomly. However, sampling cells randomly does not provide the best contrast between the positive and negative samples. The negative samples should ideally obey at least some of the statistical frequency characteristics of the positive samples, so that the factorization process can capture the more subtle (semantic) differences in the latent representation (rather than the blunt statistical differences). The first step is to pick a ratio  $\alpha > 1$  between the negative and positive samples. For each edge  $(i, j)$  in each (say  $r$ th) distance graph, a total of  $\alpha$  negative samples will be selected by repeating the randomized process below  $\alpha$  times:

- (1) For the edge  $(i, j)$  in the distance graph  $r$ , let  $Out(i)$  be the number of outgoing edges of node  $i$  and  $In(j)$  be the number of

incoming edges of node  $j$ . Between nodes  $i$  and  $j$  select one of them randomly. The probability of picking node  $i$  from the pair  $[i, j]$  is denoted by  $P[i|(i, j)]$ , and is defined as follows:

$$P[i|(i, j)] = \frac{Out(i)}{Out(i) + In(j)} \quad (3)$$

The above probability is reasonable because words with more positive links have higher chance to sample negative links.

- (2) Depending on whether  $i$  or  $j$  is selected in the previous step, one of the following two steps is executed:
  - If node  $i$  is selected, then randomly sample an outgoing “negative” cell from  $i$  in  $\mathcal{A}$ . The other end of the cell is a word not included in the current distance graph, and the probability of picking the empty cell  $(i, b, r)$  in  $\mathcal{A}$  is proportional to  $f_b^{3/4}$ , where  $f_b$  is the frequency of the  $b$ th word in the corpus. We use  $f_b^{3/4}$  because it significantly outperforms the uniform distributions [24].
  - If node  $j$  is selected, then randomly sample an incoming “negative” cell into  $j$  in  $\mathcal{A}$ . The other end of the cell is a word not included in the current distance graph, and the probability of picking the empty cell  $(b, j, r)$  in  $\mathcal{A}$  is proportional to the frequency  $f_b^{3/4}$ .

This particular way of negative sampling improves the contrast between the positive and negative cells for the sampling process. Let the cell coordinates of these negative samples be denoted by  $\mathcal{N}$ . Duplicate sampled elements in  $\mathcal{N}$  are dropped. In the common scenario where the number of sampled elements (for each distance graph) is small compared to vocabulary size, duplicate elements are relatively rare. This is because a word pair can be a duplicate only when it is sampled more than once for the same distance graph.

The cell coordinates of the positive samples  $\mathcal{P}$  are defined as all the positive entries in  $\mathcal{A}$ :

$$\mathcal{P} = \{(i, j, r) : a_{ij}^r > 0\} \quad (4)$$

It is noteworthy that the distance graph gives greater importance to precedence and ordering between words compared to *word2vec* because of the fact that the distance graph is a directed graph. This is a fair choice because the bigrams “*horse jumped*” and “*jumped horse*” do not have the same probability of occurring in the data. In comparison to the model discussed in this paper, *word2vec* treats both ends of the context window in a symmetric way in the learning process. The negative sampling procedure also takes great pains in taking care of the ordering while sampling cells. In other words, the cells  $(i, j, r)$  and  $(j, i, r)$  do not have the same probability of being sampled in the negative set  $\mathcal{N}$ . One can now set up the objective function in terms of the positive samples  $\mathcal{P}$  and negative samples  $\mathcal{N}$ . We define the *relevant set*  $\mathcal{R} = \mathcal{P} \cup \mathcal{N}$ , and optimize the objective function only over triplets  $(i, j, r)$  in  $\mathcal{R}$ :

$$\begin{aligned} J &= \frac{1}{2} \sum_{(i,j,r) \in \mathcal{R}} (a_{ij}^r - \sum_{s=1}^p [u_{is}v_{js} + u_{is}w_{rs} + v_{js}w_{rs}])^2 \\ &\quad + \gamma (\|U\|_F^2 + \|V\|_F^2 + \|W\|_F^2) / 2 \end{aligned} \quad (5)$$

where  $\|\cdot\|_F^2$  refers to the (squared) Frobenius norm, and  $\gamma$  is a scalar regularization parameter. Note that the modified objective function is not quite the same, because it downsamples the negative entries, and therefore gives greater importance to the positive entries. This

is a reasonable design choice because the positive entries do contain a lot more information for learning purposes. The parameter  $\alpha$  controls the relative weighting. This objective function can be optimized using gradient descent as

$$\begin{aligned}\frac{\partial J}{\partial u_{iq}} &= \sum_{j,r:(i,j,r) \in \mathcal{R}} e_{ij}^r(-v_{jq} - w_{rq}) + \gamma u_{iq} \forall q \\ \frac{\partial J}{\partial v_{jq}} &= \sum_{i,r:(i,j,r) \in \mathcal{R}} e_{ij}^r(-u_{iq} - w_{rq}) + \gamma v_{jq} \forall q \\ \frac{\partial J}{\partial w_{rq}} &= \sum_{i,j:(i,j,r) \in \mathcal{R}} e_{ij}^r(-u_{iq} - v_{jq}) + \gamma w_{rq} \forall q\end{aligned}$$

where  $e_{ij}^r = a_{ij}^r - \sum_{s=1}^p [u_{is}v_{js} + u_{is}w_{rs} + v_{js}w_{rs}]$ . We can then write the update equations for the matrices  $U$ ,  $V$ , and  $W$  using gradient descent with a step-size  $\eta$ . Furthermore, We can update the entire  $p$ -dimensional rows of  $U$ ,  $V$ , and  $W$  at one time, because the updates have a consistent form:

$$\begin{aligned}\bar{u}_i &\leftarrow \bar{u}_i(1 - \eta\gamma) + \eta \sum_{j,r:(i,j,r) \in \mathcal{R}} e_{ij}^r(\bar{v}_j + \bar{w}_r) \\ \bar{v}_j &\leftarrow \bar{v}_j(1 - \eta\gamma) + \eta \sum_{i,r:(i,j,r) \in \mathcal{R}} e_{ij}^r(\bar{u}_i + \bar{w}_r) \\ \bar{w}_r &\leftarrow \bar{w}_r(1 - \eta\gamma) + \eta \sum_{i,j:(i,j,r) \in \mathcal{R}} e_{ij}^r(\bar{u}_i + \bar{v}_j)\end{aligned}$$

It is also possible to use *stochastic* gradient descent (SGD) with entry-wise updates. In SGD, entry-wise updates are used, in which the entries in  $\mathcal{R}$  are sampled one by one and the errors in them are used to update the according to the following equations:

$$\begin{aligned}\bar{u}_i &\leftarrow \bar{u}_i(1 - \eta\gamma) + \eta e_{ij}^r(\bar{v}_j + \bar{w}_r) \\ \bar{v}_j &\leftarrow \bar{v}_j(1 - \eta\gamma) + \eta e_{ij}^r(\bar{u}_i + \bar{w}_r) \\ \bar{w}_r &\leftarrow \bar{w}_r(1 - \eta\gamma) + \eta e_{ij}^r(\bar{u}_i + \bar{v}_j)\end{aligned}$$

We can repeatedly cycle through all the entries in  $\mathcal{R}$  in order to perform the updates until convergence is reached. If sufficient data is available, the regularization parameter  $\gamma$  is set to 0.

### 3.1 Leveraging the Embedding

Note that the rows of both the matrices  $U$  and  $V$  yield  $p$ -dimensional word embeddings. Both embeddings store different properties corresponding to the preceding contextual words, or the following contextual words. Therefore, the  $i$ -th row of  $U$  and the  $i$ -th row of  $V$  are *concatenated* to create a  $2p$ -dimensional embedding of the  $i$ -th word. Also,  $\bar{w}_r$  is the document embedding of the  $r$ -th document.

### 3.2 Incorporating Bias Variables

It is also possible to incorporate bias variables associated with words and documents. Bias variables are particularly important when there is very wide variation in word and document frequencies. We associate the bias variable  $b_i^u$  for the outgoing edges of the  $i$ th node, the bias variable  $b_j^v$  for the incoming edges of the  $j$ th node, and the bias variable  $b_r^w$  for the  $r$ th document. The superscripts of the bias variables are chosen depending on the matrices ( $U/V/W$ ) that they logically belong to. Then, the prediction equation changes to:

$$\hat{a}_{ij}^r = b_i^u + b_j^v + b_r^w + \sum_{s=1}^p [u_{is}v_{js} + u_{is}w_{rs} + v_{js}w_{rs}]$$

The gradient descent equations remain the same except that the bias variables also need to be updated.

### 3.3 Setting the Parameters

Finally, there are three key parameters for the approach, i.e., the number of negative samples  $\alpha$ , the decay factor  $\lambda$  and the order  $k$  of the distance graph. The negative sampling parameter  $\alpha$  should always be greater than 1 to avoid trivial solution and to make the words that are not connected in distance graph to be far away. However, a too large value of  $\alpha$  will make the negative samples dominate the objective function and increase the training time. Following *word2vec* [23, 34], values in the range of 5 to 10 are reasonable for  $\alpha$ . Therefore,  $\alpha$  can be tuned in this range.

The choice of  $\lambda \in (0, 1)$  and  $k$  can also be accomplished in a similar manner. However, it does not make sense to choose this pair of parameters independently, since they are closely related. For example, if  $\lambda = 0.1$ , then any value of  $k$  above 3, will cause word pairs distant more than 3 units to have negligible weight. This suggests that it makes sense to tie the choice of these two parameters. Specifically, we allow the order of the distance graph to be a free parameter, and set  $\lambda$  as

$$\lambda = 0.1^{1/(k-1)} \quad (6)$$

This ensures that the most distant word pair has about 0.1 the weight as that of the closest word pair. The value of  $k$  is analogous to the context window used in methods like *word2vec*. Depending on the size of the data set, the value of  $k$  may range between 2 and 10. Details of how  $k$  affects performance will be given in Section 4.7.

### 3.4 Computational Complexity

It is noteworthy that each cycle of stochastic gradient descent requires time that is proportional to the number of entries in  $\mathcal{R}$ . Typically, a constant number of cycles are required, and therefore establishing the time for each cycle is useful. The number of entries in  $\mathcal{R}$  is  $(1 + \alpha)$  times the number of edges in the distance graphs,  $(1 + \alpha) \sum_{r=1}^n g_{d_r}$ , where  $g_{d_r}$  is the number of edges in the distance graph created from document  $d_r$  and  $n$  is the number of documents. Thus, the time complexity of each cycle of SGD is  $O\left((1 + \alpha) \cdot p \cdot \sum_{r=1}^n g_{d_r}\right)$ , where  $p$  is the embedding dimension. Note that the number of edges in each distance graph, i.e.,  $g_{d_r}$ , is at most  $k$  times the number of words in the document, where  $k$  is a small value that is typically less than 15. In other words, *the number of update steps for each cycle of SGD is linearly proportional to the number of non-zero entries in the document-term matrix*. The linear time complexity w.r.t corpus size can also be indirectly shown by the connection of this approach to factorization machines, which provides an alternate approach to implement it with off-the-shelf software.

### 3.5 Space Complexity

The main space complexity comes from the training data  $\mathcal{R}$  and the parameters  $U$ ,  $V$  and  $W$ , which is  $O(|\mathcal{R}| + n \cdot p \cdot d \cdot p)$ . Here  $|\mathcal{R}|$  can be calculated as  $\sum_{i=1}^n p_i \cdot (1 + \alpha)$ , where  $p_i$  is the number of links in distance graph  $A_i$  and  $\alpha$  is included because for each link, we randomly sample  $\alpha$  negative samples. Note that our framework doesn't need to store the sparse matrix  $A_i$ . It only need to store the links in  $A_i$ , which significantly reduces the space complexity. Assume that the length of the  $i$ -th document  $D_i$  is  $n_i$ , then for the extreme case, each word in  $D_i$  results in  $k$  links, where  $k$  is the

window size. Then the number of links that can be constructed using distance graph from  $D_i$  is  $O(n_i \cdot k)$ . Thus,  $p_i$  is bounded by  $n_i k$ . Then  $\sum_{i=1}^n p_i \cdot (1 + \alpha)$  is approximately  $O(k \cdot \alpha \sum_{i=1}^n n_i)$ . Therefore, the total space complexity is  $O(n \cdot p + d \cdot p + k \cdot \alpha \sum_{i=1}^n n_i)$ .

### 3.6 Connections with Factorization Machines

Since this approach uses pairwise shared matrix factorization, it is natural to connect it with factorization machines [29]. Such a conversion also suggests an efficient implementation. The basic idea is to convert the problem into a regression modeling problem by flattening the structure of the tensor into a set of rows with  $2d + n$  feature variables and a single target variable. The  $2d + n$  feature variables can be categorized into three blocks, corresponding to the *outgoing edge block* ( $d$  entries), *incoming edge block* ( $d$  entries), and *document id block* ( $n$  entries). For each entry  $(i, j, r) \in \mathcal{R}$ , the dependent variable is set to  $a_{ij}^r$ . The values in the three blocks of the feature variables are set as follows:

- (1) The  $i$ th entry of the outgoing edge block is set to 1, and all other entries are set to 0.
- (2) The  $j$ th entry of the incoming edge block is set to 1, and all the other entries are set to 0.
- (3) The  $r$ th entry of the document id block is set to 1, and all other entries are set to 0.

The gradient-descent approach [29] can be used for the factorization machine. One advantage of using a factorization machine is that efficient off-the-shelf implementations are available that require time linear in the number of entries in  $\mathcal{R}$ .

## 4 EXPERIMENTAL RESULTS

In this section, we conduct experiments to show the effectiveness of the learned word/document representation by DGTF for different tasks. Specifically, we aim to answer the following three questions:

- Are the word embedding able to capture the syntactic and semantic meanings of words?
- Can the representations learned by DGTF capture the semantic meanings of documents? and
- How good is the representation for visualization?

We begin by introducing the datasets and representative state-of-the-art word/document embedding methods. We first compare DGTF with these methods on word analogy to answer the first question. We then conduct document classification and document retrieval tasks to investigate the quality of word/document embedding, which answers the second question. Finally, we explore visualization using the embedding. Further experiments are conducted to study the sensitivity of DGTF to the hyper-parameters.

### 4.1 Datasets

Following [34], for word analogy, we train the word embedding using English WIKIPEDIA corpus [31] and test on the word analogy dataset introduced by Mikolov et al [23], which contains five types of semantic questions, and nine types of syntactic questions, with a total of 8,869 semantic and 10,675 syntactic questions.

The classification experiments are conducted on 4 publicly available benchmark datasets, which includes Economics, Market, Government and DBLP. The first three datasets, i.e., Economics, Market and Government, are three subsets extracted from RCV1<sup>2</sup>, which

<sup>2</sup><http://trec.nist.gov/data/reuters/reuters.html>

**Table 1: Statistics of Datasets for Classification**

Dataset	# instances	# classes	Doc. length
Economics	115,865	10	145.10
Market	198,060	4	119.83
DBLP	66,512	6	11.46
Government	208,486	23	169.07

**Table 2: Statistics of Dataset for Document Retrieval**

Dataset	# Documents	# queries	# Avg. Query-Doc Pairs
Ohsumed	103,445	106	177.03

is a large benchmark corpus for text classification [18]. DBLP contains titles of papers from the computer science bibliography<sup>3</sup>. We choose six diverse research fields for classification, i.e., “Artificial intelligence”, “Computer graphics”, “Computer networks”, “Database”, “High-Performance computing” and “Human computer interaction”. For each field, we select representative conferences and collect the papers published in the selected conferences as labeled documents. For the datasets Economics, Market and Government, we use both the title and first paragraph to learn the document embedding, which can be seen as long documents. For DBLP, we use only titles to learn sentence embedding, which can be treated as short texts. The statistics of the datasets are summarized in Table 1.

The document retrieval experiment is performed on Ohsumed dataset<sup>4</sup>. The Ohsumed collection is a set of 348,566 reference documents from MEDLINE, which is an online medical information database. Each document consists of title and/or abstract from 270 medical journals over a five-year period (1987-1991). We use both title and abstract to learn embedding and thus filter out documents that do not contain abstracts. We further remove references that have less than 50 words, which leaves us 103,445 references. In addition to the documents, Ohsumed also contains 163 queries. Each query has a set of documents associated with it, which forms query-doc pairs. The statistics of the dataset are shown in Table 2.

### 4.2 Compared Word/Doc Embedding Methods

We compare DGTF with other representative and state-of-the-art unsupervised word and document embedding algorithms.

- BOW: the classical “bag-of-words” represent each document as a  $M$ -dimensional vector, where  $M$  is the size of the vocabulary and weight of each dimension is calculated by the TFIDF scheme. We use it as a baseline without representation learning.
- SG: Skip-gram [23] is a popular *word2vec* model and its training objective is to find word representations that are useful for predicting the surrounding words of a selected word in a sentence. After obtaining word embedding by Skip-gram, following the common way [34, 38], we use the average word embedding of the document as the document representations, i.e.,  $\bar{d}_r = \frac{1}{n_{d_r}} \sum_{i \in d_r} \bar{u}_i$ , where  $n_{d_r}$  is the number of words in the document  $\bar{d}_r$  and  $\bar{u}_i$  is the representation of word  $i$ .
- CBOW: Continuous bag-of-words [23] is another popular *word2vec* model. Unlike Skip-gram, the training objective of CBOW is to find word representations that are useful for predicting the center word by its neighbors. Similarly, we use average word embedding to obtain the document embedding.

<sup>3</sup>Available at <http://arnetminer.org/billboard/citation>

<sup>4</sup>Available at [http://trec.nist.gov/data/t9\\_filtering.html](http://trec.nist.gov/data/t9_filtering.html)

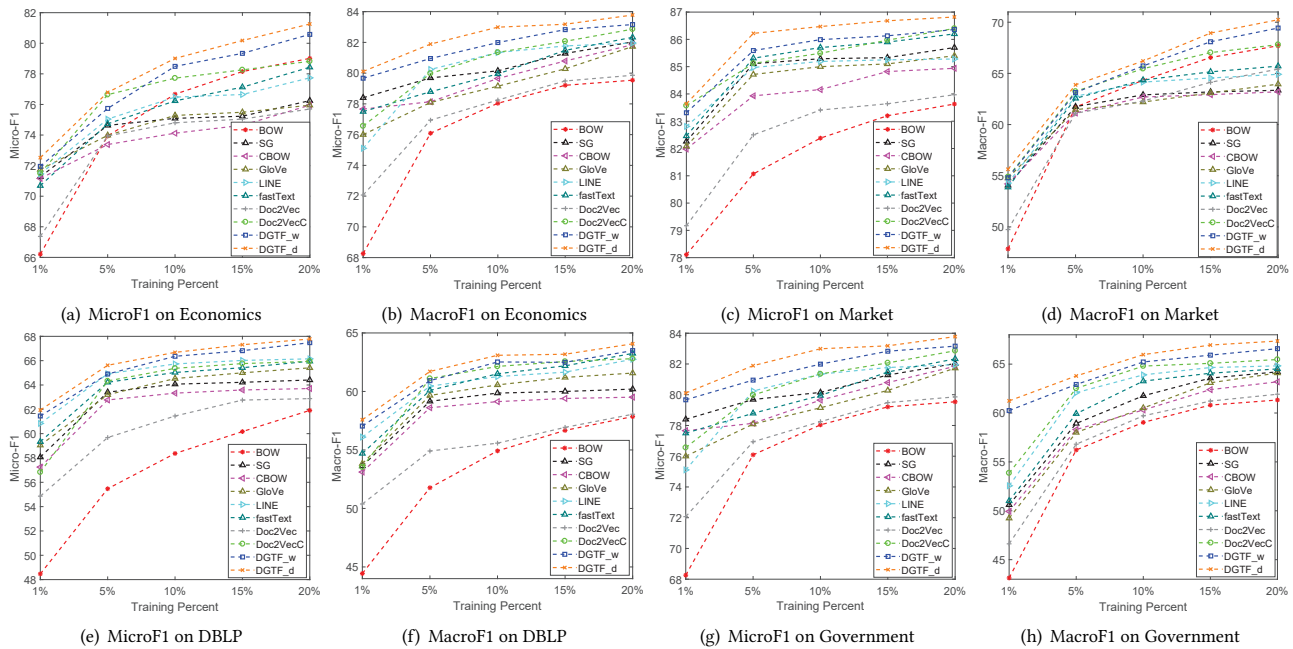


Figure 2: Document Classification Performance in terms of MicroF1 and MacroF1

- GloVe: GloVe [27] is a global logbilinear regression model performed on aggregated global word-word co-occurrence statistics from a corpus.
- Doc2Vec: This is the distributed memory version of paragraph vector [16], which aims at learning document embeddings that are good at predicting the next word given context.
- Doc2VecC [7]: This is an improved version of Doc2Vec, which introduces a data-dependent regularization to Doc2Vec that favors informative or rare words while forcing the embeddings of common and non-discriminative ones to be close to zero.
- LINE: Large-scale information network embedding model [34] is state-of-the-art word/document embedding model, which uses the word-word co-occurrence to construct the information network and then learn embedding by preserving the first-order and second-order proximity of words in the network.
- fastText [5]: This is an improved version of the Skip-gram model, where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram; words being represented as the sum of these representations.
- DGTF\_w: This is the proposed framework that uses average word embedding of DGTF to represent documents. We use this to evaluate the quality of word embedding learned by DGTF.
- DGTF\_d: This is the proposed framework which uses the document embedding of DGTF as document representation. In this paper, we focus on the document-level distance graph for DGTF and leave the sentence-level as future work.

Note that we don't compare DGTF with supervised document embedding algorithms such as PTE [33], CNN [13] and LSTM [26, 39] or linked document embedding algorithms such as LDE [38] and RTM [6] because the proposed DGTF is an *unsupervised* word and document embedding method that's designed for documents *without links*.

Table 3: Word analogy results (%) on WIKIPEDIA data

Alg.	SG	CBOW	GloVe	LINE	fastText	Doc2VecC	DGTF
Semantic	69.21	68.57	69.59	73.62	72.48	72.83	<b>74.03</b>
Syntactic	57.90	56.42	58.37	59.81	58.39	58.94	<b>60.11</b>
Overall	63.10	61.99	63.53	66.13	64.83	65.30	<b>66.48</b>

### 4.3 Word Analogy

To answer the first question, we conduct word analogy task. Given a word pair  $(w_a, w_b)$  and a word  $w_c$ , the task of word analogy is to find a word  $w_d$ , such that the relation between  $w_c$  and  $w_d$  is similar to the relation between  $w_a$  and  $w_b$  [23]. For example, given a word pair  $(France, Paris)$  and a word  $Germany$ , the right answer should be  $Berlin$  because  $Paris$  is the capital of  $France$  just as  $Berlin$  is the capital of  $Germany$ . Given the word embeddings, this task is equivalent to find the word that has the closest cosine similarity to  $\bar{w}_b - \bar{w}_a + \bar{w}_c$ , where  $\bar{w}_b$  is the embedding of  $w_b$ . Two categories of word analogy are used in this task: semantic and syntactic. The parameters of all the methods are tuned using grid search. Table 3 reports the results of word analogy trained on WIKIPEDIA. From the table, we can see that DGTF outperforms the compared methods for both semantic and syntactic tasks, which implies the effectiveness of DGTF in capturing the semantic and syntactic meaning of words.

### 4.4 Document Classification

To answer the second question, we conduct document classification to quantitatively evaluate the discriminativeness of the embedding learned by DGTF. Two widely used classification evaluation metrics, i.e., Micro-F1 and Macro-F1, are adopted to evaluate the classification results. The larger the Micro-F1 and Macro-F1 scores are, the

better the classification result is, which indicates the discriminativeness of the embeddings.

For each embedding algorithm, we first learn the representation from corpus. We then use linear SVM to perform classification with the learned representation. We fix the classifier to be linear SVM as our goal is to test the discriminativeness of the representations learned by each method, not the effectiveness of the classifier. There are some parameters to be set for the embedding learning methods and linear SVM. In the experiment, we use 10-fold cross validation on the training data to set the parameters. Note that no test labels are involved in the parameter tuning. Specifically, for DGTF, we set  $k = 5$ ,  $p = 100$ ,  $\alpha = 7$  and  $\gamma = 0.01$ . The sensitivity of parameters  $k$  and  $p$  on the effectiveness of DGTF will be analyzed in Section 4.7. We use  $x\%$  for training and the remaining  $1-x\%$  for testing. We vary  $x$  as  $\{1, 5, 10, 15, 20\}$  to have a better understanding of how DGTF behaviors under different experimental settings. Each experiment is conducted 5 times and the average performance in terms of Macro-F1 and Micro-F1 are reported in Figure 2. From the figure, we observe that:

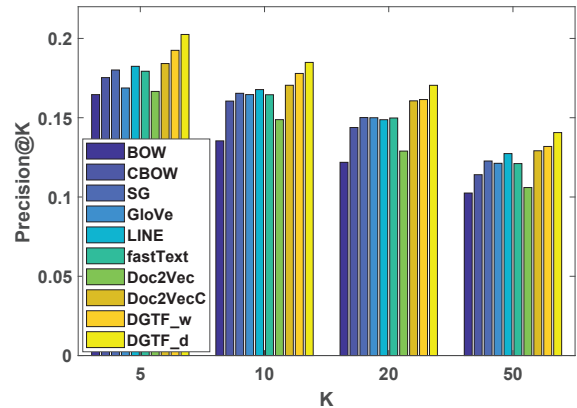
- Generally, word embedding algorithms such as SG, CBOW, GloVe and fastText outperform BOW. This shows that word/document embeddings can learn dense representations of documents which can improve the classification performances.
- The performance of LINE is better than SG and CBOW in most of the cases. LINE uses word-word co-occurrence relationship to build an information network in the corpus level, and then capture first-order and second-order proximity of words in the network, which can capture better semantic meanings of words.
- Both DGTF\_w and DGTF\_d outperforms LINE. This is because DGTF built distance graph, which assigns different weights to pair of words of different distance and thus can capture more refined word semantic meanings. In addition, LINE build information network at corpus level, which may loss a lot of relevant information in the corpus; while DGTF builds distance graph for each document and thus can keep more information.
- Both DGTF\_d and DGTF\_w outperform all the baseline methods, which shows the effectiveness of DGTF in learning word representation and document representation jointly.

#### 4.5 Document Retrieval

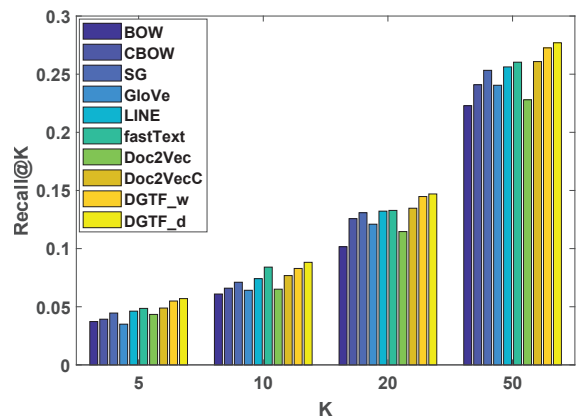
In this subsection, we further perform document retrieval using the dataset in Table 2. Document retrieval focuses on evaluating if two documents have similar semantic meanings are assigned similar vector representations, which provides another perspective in measuring the effectiveness of DGTF.

To evaluate the performance of document retrieval, we use two popular evaluation metrics, i.e., precision@k and recall@k [22]. The larger precision@k and recall@k are, the better the performance is, which indicates that documents of similar semantic meanings are assigned similar embeddings.

Note that our focus is to measure the effectiveness of the features for document retrieval, not the quality of document retrieval algorithms. Thus, for fair comparison, we simply use cosine similarity as document retrieval method. Specifically, we first learn the query and document representations using the embedding algorithms. Then given a query, we retrieve the top  $K$  documents that have the largest cosine similarity with that query. With the retrieved documents, we calculate precision@k and recall@k using the ground



(a) Precision@K



(b) Recall@K

Figure 3: Document retrieval performance on Ohsumed

truth query-doc pairs. We use grid search to tune the parameters for each representation learning algorithm. Specifically, for DGTF, we set  $k = 5$ ,  $p = 100$ ,  $\alpha = 7$  and  $\gamma = 0.01$ . Each experiment is conducted 10 times and the average performance is shown in Figure 3. From the figure, we observe that: (i) DGTF\_d and DGTF\_w outperform all the compared methods, which implies the effectiveness of DGTF by constructing distance graph for each documents and then use pairwise factorization to jointly learn word and document embedding; and (ii) DGTF\_d is slightly better than DGTF\_w, which is because DGTF\_w uses average word embedding of a document while DGTF\_d directly learn the document embedding.

#### 4.6 Visualization

One important application of document embedding is to create meaningful visualizations that layout on a two dimensional space. A good visualizations also implies the quality of the representation [32, 36, 37]. We first learn low-dimensional vector representations of the documents with different embedding approaches and then further map the low dimensional vectors to a 2D space with t-SNE [21]. Figure 4 shows the visualization of SG, LINE, Doc2VecC and DGTF\_d on the Market dataset, where each dot represent a document and the color denotes the class of the document. From the figure, we observe that: (i) The visualization of SG and Doc2VecC

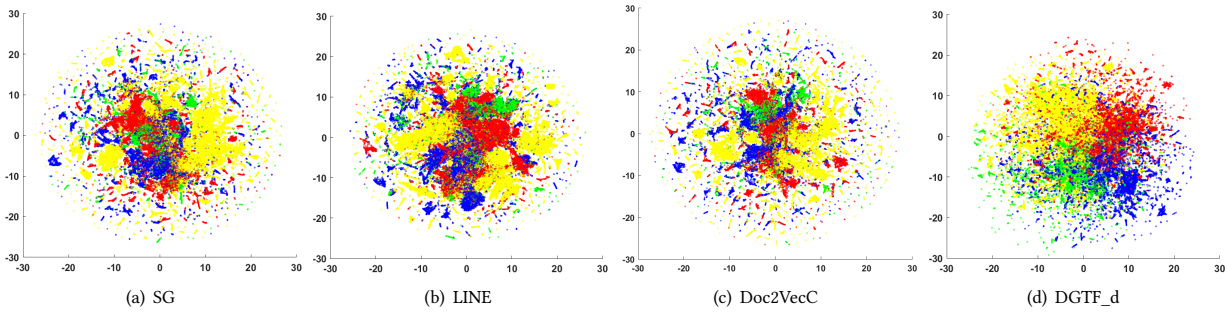


Figure 4: Visualization of Document Embedding on Market

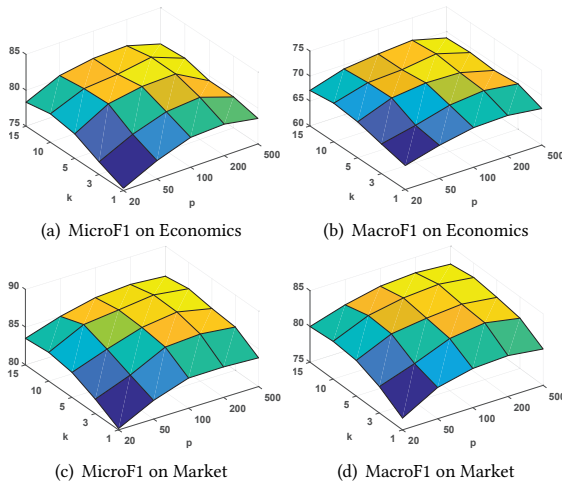


Figure 5: Parameter Analysis

are not as good as LINE because for SG and Doc2VecC, clusters belong to different classes are highly overlapped tightly into the center area; and (ii) The visualization of DGTF\_d is better than LINE because LINE split dots of the same classes into several small classes such as the yellow and blue classes; while DGTF\_d put the dots of the same classes into the same large cluster. This suggests that the embedding learned by DGTF can help visualization, which implies the effectiveness of the representation learned by DGTF.

#### 4.7 Parameter Analysis

The proposed framework has two important parameters,  $k$  and  $p$ , where  $k$  defines the order of the distance graph and controls the value of  $\lambda$  via Eq.(6), and  $p$  is the embedding dimension. In this section, we investigate the impact of  $k$  and  $p$  on the performance of DGTF. We only show results on Economics 20% and Market 20% for document classification since we have similar observations with other experimental settings. We empirically set  $\alpha = 7$  and  $\gamma = 0.01$ . We vary the values of  $k$  as  $\{1, 3, 5, 10, 15\}$  and the values of  $p$  as  $\{20, 50, 100, 200, 500\}$ . Every experiments are run 5 times and the results are shown in Figure 5. It can be observed from the figure that: (i) Generally, with the increment of  $k$ , the performance tends to first increase then become stable. This is because when  $k$  is small, the distance graph is too sparse, which cannot well capture the words relationships. Similarly, when we increase  $p$  from 20,

the performance also increases as a larger embedding provides for representation capacity. However, a too large  $p$  may deteriorate the performance as the model becomes too complex and also increase the training time; and (ii) In general, when  $k$  is within  $[5, 10]$  and  $p$  is within  $[100, 200]$ , we can achieve a relatively good performance, which eases the parameter selection.

### 5 RELATED WORK

Word/Document embedding is an important research topic which has attracted increasing attention [3–5, 19, 24, 25, 27, 28, 35]. It learns low dimensional vector representations of words to capture words’ semantic and syntactic meanings. The resulting representation has various applications such as word analogy [24], POS tagging [9], document classification [33] and information retrieval [8].

Various word embedding approaches has been proposed, which mainly follows the distributional hypothesis idea [10] that “you shall know a word by the company it keeps”. Following such idea, word-word co-occurrence relationship are extracted from sentences and texts, and word embedding are trained to preserve such co-occurrence relationship. For example, Skip-gram [24] learns word representation that is good at predicting nearby words. fastText [5] extends Skip-gram by learning representations for character n-grams, and to represent words as the sum of the n-gram vectors. GloVe [27] adopts a global logbilinear regression model performed on aggregated global word-word co-occurrence statistics from a corpus to learn word embedding. LINE [34] represent the word-word co-occurrence relationship as an information network and performances network embedding to learn word representations. It is found that the aforementioned approaches are actually matrix factorization on word-word co-occurrence matrix [17]. Bamler et al. [3] further investigated dynamic word embedding by extending a probabilistic version of Skip-gram to capture the connection of embeddings between constitutive time-stamps. Following a similar idea of word embedding, document embedding (doc2vec) are proposed, which tries to learn word and document representations that capture the word-document relationship. For example, paragraph vector [16], extends Skip-gram by learning document embeddings that are good at predicting the next word given context. Doc2VecC [7] introduces a data-dependent regularization to paragraph vector that favors informative or rare words while forcing the embeddings of common and non-discriminative ones to be close to zero. LDE [38] studied document embedding for document network by taking document dependency into consideration. PTE [33] investigated document embedding for classification.

Our approach also investigates the word-word relationship for word and document embedding. However, our framework is inherently different from existing work: (i) we provide a principled approach for joint word and document embedding by exploring the use of distance graph, which can capture word relations in a more refined fashion; while exiting work focus on word or document embedding only and doesn't consider the difference of preceding and following words; and (ii) the majority of exiting methods are actually matrix factorization on word-word co-occurrence matrix [17]; while we adopt tensor factorization and propose an efficient way to simultaneously learn word and document embedding.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper, we explore the use of a distance graph and negative sampling in order to capture the word relationship in a more fine-grained fashion. We proposed a novel framework of tensor factorization on document graphs for word and document embedding and provide an efficient implementation which converts tensor factorization to pairwise shared matrix factorization. Experimental results on real-world datasets demonstrated the effectiveness of the embedding learned by DGTF for various tasks such as document classification, document retrieval and visualization.

There are several interesting directions that need further investigation. First, in this work, we focus on document-level distance graph. As future work, we would like to investigate the performance of sentence-level and paragraph-level distance graph. Second, linked documents are very pervasive in social media [38]. Therefore, we also want to extend DGTF to handle documents with link information.

## ACKNOWLEDGMENTS

This material is based upon work supported by, or in part by, the National Science Foundation (NSF) under grants #1614576 and #1610282, and the Office of Naval Research (ONR) under grant N00014-17-1-2605.

## REFERENCES

- [1] Charu C Aggarwal and ChengXiang Zhai. 2012. *Mining text data*. Springer Science & Business Media.
- [2] Charu C Aggarwal and Peixiang Zhao. 2013. Towards graphical models for text processing. *Knowledge and information systems* 36, 1 (2013), 1–21.
- [3] Robert Bamler and Stephan Mandt. 2017. Dynamic Word Embeddings. In *ICML*. 380–389.
- [4] Ghazaleh Beigi, Kai Shu, Ruocheng Guo, Suhang Wang, and Huan Liu. 2019. I Am Not What I Write: Privacy Preserving Text Representation Learning. *arXiv preprint arXiv:1907.03189* (2019).
- [5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.
- [6] Jonathan Chang and David Blei. 2009. Relational topic models for document networks. In *Artificial Intelligence and Statistics*. 81–88.
- [7] Minmin Chen. 2017. Efficient vector representation for documents through corruption. In *ICLR*.
- [8] Stéphane Clinchant and Florent Perronnin. 2013. Aggregating continuous word embeddings for information retrieval. In *CVSC*. 100–109.
- [9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR* 12, Aug (2011), 2493–2537.
- [10] John R Firth. 1957. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis* (1957).
- [11] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2014).
- [12] Parry Husbands, Horst Simon, and Chris HQ Ding. 2001. On the use of the singular value decomposition for text retrieval. *Computational information retrieval* 5 (2001), 145–156.
- [13] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *ENBLP*.
- [14] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.
- [15] Thomas K Landauer. 2006. *Latent semantic analysis*. Wiley Online Library.
- [16] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*. 1188–1196.
- [17] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.
- [18] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *JMLR* 5 (2004), 361–397.
- [19] Yang Li, Quan Pan, Tao Yang, Suhang Wang, Jiliang Tang, and Erik Cambria. 2017. Learning word representations for sentiment analysis. *Cognitive Computation* 9, 6 (2017), 843–851.
- [20] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *JMLR* 2 (2002), 419–444.
- [21] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* 9, Nov (2008), 2579–2605.
- [22] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press.
- [23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR Workshop*.
- [24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.
- [25] Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*. 2265–2273.
- [26] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *TASLP* 24, 4 (2016), 694–707.
- [27] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. 1532–1543.
- [28] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.
- [29] Steffen Rendle. 2010. Factorization machines. In *ICDM*. IEEE, 995–1000.
- [30] Steffen Rendle and Lars Schmidt-Thieme. 2010. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*. ACM, 81–90.
- [31] Cyrus Shaoul. 2010. The westbury lab wikipedia corpus. *Edmonton, AB: University of Alberta* (2010).
- [32] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. 2016. Visualizing large-scale and high-dimensional data. In *WWW*. 287–297.
- [33] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *SIGKDD*. ACM, 1165–1174.
- [34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
- [35] Luke Vilnis and Andrew McCallum. 2014. Word representations via gaussian embedding. *arXiv preprint arXiv:1412.6623* (2014).
- [36] Suhang Wang, Charu Aggarwal, Jiliang Tang, and Huan Liu. 2017. Attributed signed network embedding. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 137–146.
- [37] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed network embedding in social media. In *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 327–335.
- [38] Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. 2016. Linked document embedding for classification. In *CIKM*. ACM, 115–124.
- [39] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *NAACL*.