When Word Embedding Meets Lexical Networks

Research Project for CS512: Data Mining Principles

Liang-Wei Chen  
University of Illinois at Urbana-Champaign  
lchen112@illinois.edu

Wei-Chun Lee  
University of Illinois at Urbana-Champaign  
wlee114@illinois.edu

Hsiang-Wei Hwang  
University of Illinois at Urbana-Champaign  
hhwang2@illinois.edu

ABSTRACT

Lexical networks such as WordNet or BabelNet contain wealthy structural information. On the other hand, distributional word representations encode the semantic information and can be generally used for many Natural Language Processing (NLP) tasks. In this paper, we propose a framework that learns embeddings from lexical networks. We also introduce novel enhancements that improve the quality of our embeddings. Experimental results demonstrate that our proposed approach performs effectively on the benchmark datasets.

KEYWORDS

word embedding, entity-relation learning, lexical network, natural language processing, machine learning

1 INTRODUCTION

The word embedding techniques that learn distributional (semantic) word representations are recently popular. They have been applied to many NLP tasks, such as word sense disambiguation, similarity, or knowledge representation tasks. Corpus-based word embeddings, e.g. Word2vec [16] and Stanford Global Vectors for Word Representation (GloVe) [20], have reached promising results as word-level features.

However, as Iacobacci, Pilehvar, andNavigli [10] point out, the above corpus-based word embeddings have two major limitations: First, they ignore the fact that a word may have multiple senses. For example, a word bank may refer to a financial institution or a river bank, but Word2vec and GloVe learn all different senses into the same vector representation. Second, those corpus-based word embeddings learn from raw corpus but ignore the rich information provided by existing structured resources. Lexical knowledge networks such as WordNet [17] or BabelNet [19], should contain well-structured resources that can be used to learn the embeddings.

Starting from this motivation, the objective of our project is to learn effective word embeddings from lexical networks by exploiting the multi-sense information. First, we learn sense embeddings from WordNet by entity-relation embedding algorithms TransE [4] or TransD [12], where each sense in WordNet is an entity and each relation is a link between two senses (See Figure 1). However, learning from only lexical network relations ignores the rich co-occurrence information in corpus-based embeddings, and the Wordnet relations are sparse with average degree $\approx 2.77$, and $x$. Thus, we propose three methods to improve the quality of our sense embeddings:

1. We initialize our sense embedding which incorporates co-occurrence information from the corpus.
2. We regularize the sense embeddings to ensure the sense embeddings do not overfit the sparse relations and go far from the pre-trained word embedding.
3. We augment the relations from additional knowledge bases, e.g. BabelNet, and increase the average degree to 6.2.

Then, we transform those sense embeddings back to word embeddings by uniformly averaging or weighted averaging the sense vectors. This process makes our embeddings able to be used in many further NLP tasks. Finally, we examine the quality of our embeddings by running experiments on word embedding benchmark datasets. Our approach outperforms state-of-the-art word embedding methods.

The rest of the paper is organized as follows: In Section 2 we introduce related works. In Section 3, we discuss notations and some background knowledge. Section 4 describes our framework and the implementation details of our method. In Section 5, we show and discuss the results on the benchmark datasets. Finally, in Section 6 we conclude our analysis of the task.

2 RELATED WORK

As the bank example introduced in Section 1, embedding multi-sense information should be beneficial for embeddings. In this section, we introduce related works in leverage sense embeddings.

Chen et al. [6] introduce an idea that word sense representation and word sense disambiguation can benefit from each other. To explore this idea, they first use Skip-gram to train the word embeddings as the initialization. Then, they get the sense embeddings based on the descriptions and examples of the senses in WordNet (See fig. 1). For example, the sense embedding of bank$_{s1}$ should be composed by the description "sloping land (especially the slope beside a body of water)" and the two examples. Then, they average of the word embeddings within the sense description and examples to get the sense embedding.

Rothe and Schütze [22] (AutoExtend) extend the standard embeddings for word senses and lexeme in WordNet. They introduce two constraints: (1)senses are sums of their lexemes, and (2) words are sums of their lexemes. The first one is similar to what [6] has introduced. As for the second constraints, they point on the relation between sense embeddings and word embeddings. For instance, the word "bloom" contains two senses bloom(organ) and bloom(period), so the embedding of the word "bloom" is a sum of the embeddings of bloom(organ) and bloom(period).

Iacobacci, Pilehvar, andNavigli [10] (SensEmbed) also extend the standard word embedding by complementary knowledge from structured sources. They first generating a sense-annotated corpus
by disambiguating words to its BabelNet senses with Babelfy. [18].
After the disambiguation, each word will be replaced by the their corresponding senses in the corpus. Then, they apply Word2vec to produce continuous representations of word senses. They focus on different strategies to calculate word similarity pairwisely by sense embeddings but they have no word embeddings. Indeed, the above embedding algorithms are able to capture sense information and tackle tasks like word similarity and word sense disambiguation. However, they ignore the rich relation information from connections in WordNet and BabelNet. The algorithms consider WordNet and BabelNet as supplemental tools when conducting the sense embedding mining. It inspires us to access more information in the lexical networks to create informative embeddings.

3 BACKGROUND KNOWLEDGE
In this section, we introduce some background knowledge and define notations. First of all, we define notations that we are going to use throughout the paper. Second, we introduce different approaches in entity-relation embeddings (or knowledge base embeddings). Third, we discuss WordNet, a lexical network that we use in the project.

3.1 Notations
We first introduce some notations. We follow the notations in the previous work [12]. We denote the set of entities as \( E \subseteq \mathcal{R}^N \) and the set of relations as \( R \subseteq \mathcal{R}^M \). The size of \( E \) is \( n_e \) and the size of \( R \) is \( n_r \). Every relation triplet consists of two entities and one relation. We represent each triplet in the form \((h, r, t)\) where \( h \) and \( t \) represent entities and \( r \) represents relations. For example, \((\text{waterside}, \text{hyponym}, \text{Bank(geographical)})\) is a relation triplet. We use bold letters to represent vectors. For example \( h \) is the distributed representation of entity \( h \).

For those triplets that are truths, we call them golden triplets. We denote the set of golden triplets as \( \Delta \). We also use \( \Delta' \) to represent the set of triplets that are not true, which we call corrupt triplets. In the context of network embedding, researchers usually want to know whether a triplet is a golden triplet. And one approach is to design a score function that can rank all the triplets with respect to one relation. We denote such score function as \( f_r(h, t) \). We want to design a score function so that golden triplets have higher scores while other triplets have lower scores.

3.2 Translating Embeddings for Modeling Multi-relational Data
We discuss TransE and the works based on TransE in this subsection. The major difference between these models is the score function they use. We also present the comparison in table 1.

TransE. Inspired by word2vec [16], Bordes et al. [4] proposed TransE to embed both entities and relations in knowledge base into low dimensional vector space. Their goal is to model relations as translations of entity vectors. That is \( h + r \approx t \). Therefore, Bordes et al. proposed the following score function:

\[
 f_r(h, t) = -\|h + r - t\| \tag{1}
\]

Where norm can be either L1-norm or L2-norm. The score function should have high value when the given triplet is a golden triplet, and the score should be small when the given triplet is a corrupt triplet.

In order to capture the desired behavior for score function, Bordes et al. proposed TransE to maximize the following margin-based objective function:

\[
 \mathcal{L} = \sum_{(h, r, t) \in \Delta} \sum_{(h', r, t') \in \Delta'} \left[y + f_r(h, t) - f_r(h', t')\right]_+ \tag{2}
\]

Notice that the \([x]_+\) represents \(\max(x, 0)\).

We present the framework for TransE [4], and also other works in TransX series, in Algorithm 1.

TransH, TransR, and TransD. TransE performs well in link prediction work, but their model cannot capture complicated relations like reflective relations, one-to-many relations, many-to-one relations, and many-to-many relations [25]. Wang et al. [25] observed such phenomenon and propose to model a relation as an translation of entities on a hyperplane. More specifically, they characterized the hyperplane with its normal vector \( w_r \in \mathcal{R}^N \). The projection of an entity \( e \) can therefore be computed by the following formula:

\[
 e_\perp = e - w_r^T e w_r \tag{3}
\]
And they define their score function as follows:

\[ f_r(h, t) = -\|h + r - t\|_2^2 \] (4)

The intuition of their approach is that in different relations, entities are projected to different translation space. This allows entities to perform differently in different relations.

Lin et al. extended TransH by embedding entities and relations in different vector space. Recall that TransH project an entity to an hyperplane. But the project is still in the same dimensional vector space so both \(e\) and \(e_\perp\) are in \(\mathbb{R}^N\). Lin et al. relax this constraint and embed relations into different dimension vector space \(\mathbb{R}^N\). To do this, they proposed a projection matrix \(M_r \in \mathbb{R}^{N \times M}\). The projection of each entity \(e\) can be calculated as follows.

\[ e_\perp = M_r e \] (5)

The score function they propose ins the same as eq. (3).

TransR model is more expressive than previous model. However, their score function involves computing matrix-vector multiplication and therefore suffers from efficiency issue.

Based on TransR, Ji et al. [12] proposed TransD to dynamically determine the mapping matrix for different entity-relation combinations. Their motivation is to construct a more expressive model than TransR and remove the matrix-vector multiplication in the same time.

More specifically, for each entity \(e \in \mathbb{E}\), there are two vectors \(e \in \mathbb{R}^N\) and \(e^p \in \mathbb{R}^N\). \(e\) represents the meaning of the entity while \(e^p \in \mathbb{R}^N\) is used for constructing mapping matrix. Each relation \(r \in \mathbb{R}\) is also represented by two vectors \(r \in \mathbb{R}^M\) and \(r^p \in \mathbb{R}^M\).

To construct projection matrix, given a triplet \((h, r, t)\) and all the associated vectors, two matrices are constructed: \(M_{rh}, M_{rt} \in \mathbb{R}^{M \times N}\). We can compute these matrices by the following formula:

\[ M_{rh} = r^p h_p^T + 1^{M \times N} M_{rt} = r_p t^T + 1^{M \times N} \] (6)

Notice that \(I\) is the identity matrix. Given the matrices, the projection of head entity \(h\) and tail entity \(t\) can be computed by matrix-vector multiplication. However, by construction, the matrix-vector multiplication part can be replaced by vector operation. Take head entity for example,

\[ h_\perp = M_{rh} h = h_p^T r_p + [h^T, 0]^T \] (7)

This is a lot more efficient that matrix-vector multiplication.

TransD is more expressive because head and tail entity are mapped by a different matrix and also keep it efficiency.

In this work, we choose to use only TransE and TransD. This is because of two reasons. First of all, we choose TransE because it is a simple and efficient model. With less complexity, it may generalize better than complicated models. Second, we choose TransD because it is the most expressive model in the series and is also more efficient than TransR.

### Table 1: Score functions used for different embedding models. [14]

<table>
<thead>
<tr>
<th>Model</th>
<th>score function (f_r(h, t))</th>
<th>Number of parameters</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransE</td>
<td>(-|h + r - t|_2)</td>
<td>(O(n_e N + n_r N))</td>
<td>Entities and relations are in the same vector space.</td>
</tr>
<tr>
<td>TransH</td>
<td>(-|h - w_r^T h w_r - (t - w_t^T t w_t^T)|_2)</td>
<td>(O(n_e N + n_r N))</td>
<td>Entities and relations are in the same vector space.</td>
</tr>
<tr>
<td>TransR</td>
<td>(-|M_r e + r - M_t e|_2)^2</td>
<td>(O(n_e N + n_r M + n_r M))</td>
<td>Entities and relations are in the different vector spaces.</td>
</tr>
<tr>
<td>TransD</td>
<td>(-|M_{rh} e + r - M_{rt} e|_2)^2</td>
<td>(O(n_e N + n_r M))</td>
<td>Entities and relations are in the different vector spaces.</td>
</tr>
</tbody>
</table>

#### Algorithm 1: TransX framework recalled from [4]

**Data:** Training set \(S = (h, r, t)\), entities \(E\), relations \(R\), margin \(\gamma\), embedding dimension \(k\)

1. for entity \(e \in E, r \in R\) do
   2. \(r \leftarrow \text{InitializeEmbedding}()\)
   3. \(e \leftarrow \text{InitializeEmbedding}()\)
4. repeat
   5. for each entity \(e \in E\) do
      6. \(S_{\text{batch}} \leftarrow \text{sample}(S, b)\) \(\quad /\*\) sample a minibatch of size \(b\) \(*\)
      7. \(T_{\text{batch}} \leftarrow \emptyset\) \(\quad /\*\) initialize the set of pairs of triplets for training \(*\)
   8. for \((h, r, t) \in S_{\text{batch}}\) do
      9. \(S'_{(h,r,t)} \leftarrow \text{set of corrupted triplet}\)
     10. \((h', r', t') \leftarrow \text{sample from } S'_{(h,r,t)}\)
     11. \(T_{\text{batch}} \leftarrow T_{\text{batch}} \cup \{(h, r, t), (h', r', t')\}\)
   12. Update embeddings based on
        \[ \sum_{(h, r, t) \in T_{\text{batch}}} \gamma + f_r(h, t) - f_r(h', t') \] +
4. until loss value is smaller than a threshold

#### 3.3 WordNet

We use WordNet [17] to generate the connections among the senses. WordNet contains over 100,000 senses and 300,000 edges. It bases the information on the dictionary and syntactic analysis. WordNet can respond according to the input word and generate possible sets of senses in the network. For example, the word bank has different meanings. In a syntactic way, bank can be a noun and a verb. In a semantic way, it represents different meanings such as riverbank, financial institution, and pile. A word may have 9.5 senses on average in WordNet. These senses are basic units in lexical networks. We will explain how we build the lexicon and obtain the senses and relations in Section 5. The average degree for each sense is 2.77 in our experiments.

Besides, the senses connect to other senses by different relation edges (See Figure 1). There are twenty-two types of relations...
in WordNet. They reveal different semantic meanings between words. For example, *(bank.v.06, Synonyms, deposit.v.01)* means that deposit.v.01 is a synonym of bank.v.06. *(bank.v.06, Antonyms, deposit.v.01)* means that deposit.v.01 is an antonym of bank.v.06. These connections explicitly describe the relations between words and are unavailable by corpus-based methods which collect information from the corpus by a sliding window.

### 4 OUR FRAMEWORK

We want to learn high-quality sense embeddings to generate word embeddings. There are two major challenges in learning the sense embeddings from WordNet. First of all, WordNet does not contain any corpus co-occurrence information due to its nature. The second challenge is the sparsity of the WordNet. As mentioned in the previous section, the average degree of WordNet is only 2.77 while they have than 76,641 entities.

We design two methods to tackle these challenges. First, we leverage existing word embeddings to enhance sense embeddings. This allows our sense embeddings to learn co-occurrence information. Second, we augment WordNet by adding relations in BabelNet. The framework of our approach is presented in Figure 2.

The rest of the section is organized as follows. We first discuss how we apply translating embedding-based framework in our task. And then introduce how we enhance sense embeddings with existing word embeddings. We then present how we augment WordNet by adding relations in BabelNet. We also briefly introduce BabelNet and its importance in our approach. In the end of the section, we present our method to convert sense embeddings to word embeddings.

#### 4.1 Learning from WordNet

We want to learn relations between sense from structured information networks. The major challenge is how to capture relations structures in sense embeddings. Translating embedding algorithms are one of the most promising embedding frameworks that can learn from multi-relational data [4]. So we apply translating embeddings in our work.

As mentioned in the background knowledge, translating embedding aims at modeling relations as translations of entities. This facilitates our goal in learning sense embedding from WordNet. For example, *(waterside, hyponym, Bank*(geographical)*)) is a triplet in WordNet. As illustrated in Figure 3, by learning from this relationship, translating embedding will try to separate bank*(geographical)* and bank*(financial)* in order to capture this kind of relationship.

We observe that by simply using translating embeddings, the performance of embeddings at word level is poor (see Table 2). As mentioned in the beginning of the section, we suspect the reasons are the sparsity of the network and the lack of co-occurrence information. In the next subsection, we propose GloVe enhancement to strengthen sense embeddings.

#### 4.2 GloVe Enhancement: Initialization and Regularization

We propose two enhancements to incorporate co-occurrence information: GloVe initialization and GloVe regularization.

Inspired by Long et al. [15], we propose to use GloVe word embeddings to initialize sense embeddings. Long et al. [15] point out that using word embedding as initialization in TransE leads to faster convergence and better performances on the link prediction task. However, they simply average the words in the description to obtain the initialization, which might conflate unrelated words, e.g. stop words. Instead, we propose to initialize the sense embedding considering only directly related words. For example, we use the embedding of the words *bank* and *banks* to initialize all of their sense embeddings like *bank*(geographical) and *bank*(financial). We expect to see better sense embeddings so that we can obtain effective word embeddings. We choose to use GloVe [20] as a starting point. It is one of the state-of-art word embedding techniques. We use the the version learned from Wikipedia 2014 and Gigaword 5.

However, different words may have the overlapping set of senses, and we obtain the initialization for a sense by uniformly averaging all the word embeddings corresponding to this sense. For example, given a sense *bank* and two distinct words *bank* and *banked* with two pre-trained word embeddings, the word *bank* has senses *(bank, bank, . . . , bank*, bank*, bank*, bank*, . . .)* and the *banked* has the senses *(bank*, bank*, . . . , bank*, bank*, . . .)* in WordNet. The initialization of *bank* will be the average of the two word embeddings *bank* and *banked*.

There are two benefits of this approach. The first one is that we are able to add in co-occurrence information. Researchers have proposed several types of word embedding techniques which are able to capture co-occurrence information and perform really well in many NLP tasks. By using word representation as initialization, our initial sense embeddings capture co-occurrence information at word-level.

Another challenge arise after we apply GloVe initialization. During the training, initialized sense embeddings may move away from the starting point. And the embeddings of the senses may potentially lose co-occurrence information. We propose to use regularization to solve this issue.

There are two benefits of this approach. As illustrated in Figure 4(c), we add regularization to restrain the distance between learned sense embeddings and word embeddings. Adding regularization can prevent sense embeddings to be too different from word embeddings and loss the co-occurrence information inside it. Another benefit of regularization is that it can potentially help the sparsity of knowledge base network. For example, the average degree in WordNet is around 2.77 which is very sparse. The sparsity may results in over-fitting because there may not be many relations to learn from.

We use the framework in trans-series methods (Algorithm 1) but we propose a new loss function. Given a sense embedding *h*, we denote *h* as the word representation of the surface word of the sense. The new loss function is as follows.

\[
L = \sum_{(h, t) \in A} \sum_{(h', t') \in A'} [y + f_z(h, t) - f_z(h', t')]_{+} \\
+ C \left( \|h - h_l\|^2_2 + \|t - t_l\|^2_2 + \|h' - h_l'\|^2_2 + \|t' - t_l'\|^2_2 \right) \quad (8)
\]

1https://nlp.stanford.edu/projects/glove/
4.3 Relation Augmentation

Another way to resolve sparsity issue is to simply add more relations. Our framework can do this in two ways. First, we can directly combine to method. We call this off-line method. The second way is to do it in an on-line manner. For example, after learning from WordNet, we will generate a set of sense embeddings. We can use it as a starting point and learn from other knowledge bases like BabelNet. This makes our learning framework very flexible. Our approach does not have to gather all the information and learn everything at once. The results of our baseline version and on-line version are presented in section 5.

To conduct relation augmentation, we generate additional information from BabelNet. BabelNet contains trillions of entities and 38 types of connections. It inherits all entities and relations in WordNet and extends the lexical network by accumulating more entities and edges from Wikipedia database. Because of the manually edited characteristic in Wikipedia, BabelNet consists of many implicit relations and new information. For example, society\textsubscript{n} means a group of people having a distinctive cultural and economic organization, but there’s no direct relation between society\textsubscript{n} and people\textsubscript{n}. By combining the relations from BabelNet, the triplet \texttt{(society.n.01, wikidata:hyponym, people.n.01)} appears in our training data.

Thus, we can retrieve more information from BabelNet which contains all entities in WordNet and much more edges. We originally have 76,641 entities and retrieve 212,286 edges from WordNet. After mapping the entities to BabelNet, we extract 263,245 additional edges from BabelNet for the relation augmentation.

4.4 From sense embedding to word embedding

Our ultimate goal is to have quality word embeddings. Iacobacci, Pilehvar, and Navigli [10] learn their sense embeddings, but they don’t have the word-level representation. They calculate the word-level similarity by iterating over all senses of the words pairwisely. Thus, following the assumption in [22], we make an assumption that each word embedding is the linear combination of their sense embeddings. We describe two strategies to linearly combine those sense embeddings.

1. Uniform Average: For each word in our lexicon, we query WordNet to get its senses. Then, we simply average over the sense embedding vectors to be the word embeddings. For example, given an input word \textit{bank}, its word embedding it the average of senses \textit{bank}\textsubscript{n}, \textit{bank}\textsubscript{2}, \textit{bank}\textsubscript{3}, ...

2. Weighted Average: Instead of uniformly average over the sense embedding vectors, we weighted average the sense embedding vectors by its priors. The prior of a sense is obtained by summing over its lexeme counts in WordNet.

Our combination methods above generate a general word-level representation from sense embeddings, and the embeddings can be applied on many further NLP tasks.

5 EXPERIMENTS

In this section, we describe our experiments. To collect the senses in WordNet, we download pre-trained GloVe learned from Wikipedia
2014 and Gigaword 5 \(^2\). We collect our lexicon by querying WordNet to get all its senses and the relations between those senses. There are 76,641 corresponding senses and 212,286 relation triplets in WordNet and the average degree of each sense is 2.77 Each triplet contains a head sense, a tail sense, and the relation. These triplets are required information to conduct TransE and its follow-up enhancements. We only keep word has at least one sense in our lexicon.

5.1 Evaluation on word embeddings
We follow the descriptions and datasets that are introduced in [2, 11, 24]. \(^3\) to perform 3 tasks.

(1) Relatedness and Similarity: Given a pair of words, the similarity of the two embeddings should have a high correlation with their human annotated scores, where the correlation is measured by Spearman correlation. We evaluate our word embeddings on standard word similarity and relatedness datasets: MEN [5], RG65 [23], SimLex999 [9], Amazon Mechanical Turk (MTurk) [21], and WS353 [8].

(2) Analogy: Given a term x and a relation a:b, the goal is to find a term y so that x:y have the same relation as a:b. For example, given a term king and relation man:woman, the term queen should be the target term. We use the dataset in SemEval2012 task2 (SemEval2012-2) [13]. This task recognizes the continuous range of degrees of relational similarity between a given reference word pair and other pairs, which is also measured by Spearman correlation.

(3) Categorization: Given a set of concepts, this task is to cluster them into categories. For example, cats and giraffe should be clustered into the mammal class, while lizards and dinosaurs should be clustered into the reptile class.

We compare our approach with two baselines: GloVe and SensEmbed [10]. We download the pre-trained and GloVe word vectors and sense embedding vectors of SensEmbed\(^4\), and we keep only the words in our lexicon. For the sense embedding, we uniformly average them to obtain the word embeddings. Iacobacci, Pilehvar, and Navigli [10] provide only sense embeddings but calculate their word-level similarity in a pairwise manner, which lacks the intermediate word embedding and is difficult to applied on other NLP tasks. They also need to calculate similarity online between every pair of words over all their senses with quadratic run time complexity. As a result, we think it’s more reasonable to evaluate their embeddings in the word-level.

Table 2 shows our end-system performance comparing with those two baselines, and the word embeddings in this table are all obtained by the uniform average strategy. The experiment results demonstrate that our approach outperforms those baselines in most of the benchmark datasets. The GloVe initialization plays an important role while learning. The relation augmentation and the GloVe regularization also improve the performances from the pure transE. Also, the overall framework has the best performance with transE instead of transD.

In Table 4 we compare the uniform averaging strategy and the weighted averaging strategy that transforms sense embeddings into word embeddings. Surprisingly, the uniform strategy already reaches comparable performances. The weight for a sense is obtained by summing over its lexeme counts in WordNet, but there are 90,469 out of total 117,659 that are zeros. Using those priors will inevitably eliminate the information from those low-frequency senses. There, most of the scores we report using the uniform averaging strategy. Therefore, following this observation, the experiments below shows the results of the uniform averaging strategy.

Table 5 shows the performance with the different corpus embedding rate\(C\). We conduct the experiment by adding the pure transE with the Glove enhancement. Interesting, pure transE shows comparable performances on the relatedness tasks, but in the analogy tasks and the categorization tasks, the GloVe enhancement generally helps with good parameters (\(C = 0.001\) and \(C = 0.0001\)).

We also compare the transE with transD with or without the relation augmentation. It’s worthy to highlight that while the relation

\(^2\)https://nlp.stanford.edu/projects/glove/

\(^3\)We use the evaluation scripts in the Github repository: https://github.com/kudkudak/word-embeddings-benchmarks

\(^4\)We download SensEmbed vector from http://lcl.uniroma1.it/sensembed/
<table>
<thead>
<tr>
<th>Relatedness (Similarity)</th>
<th>Analogy</th>
<th>Categorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEN</td>
<td>0.7375</td>
<td>0.1660</td>
</tr>
<tr>
<td>MTurk</td>
<td>0.6300</td>
<td>0.6368</td>
</tr>
<tr>
<td>RG65</td>
<td>0.7695</td>
<td>0.8200</td>
</tr>
<tr>
<td>SimLex999</td>
<td>0.3705</td>
<td>0.3650</td>
</tr>
<tr>
<td>WS353</td>
<td>0.5166</td>
<td></td>
</tr>
<tr>
<td>SemEval2012_2</td>
<td>0.0104</td>
<td>0.5074</td>
</tr>
<tr>
<td>AP</td>
<td>0.1021</td>
<td>0.3650</td>
</tr>
<tr>
<td>BLESS</td>
<td>0.8200</td>
<td></td>
</tr>
<tr>
<td>GloVe (baseline)</td>
<td>0.5898</td>
<td>0.5166</td>
</tr>
<tr>
<td>SensEmbed (baseline)</td>
<td>0.4064</td>
<td>0.1378</td>
</tr>
<tr>
<td>transE (pure)</td>
<td>0.3024</td>
<td>0.1015</td>
</tr>
<tr>
<td>transE + regularization</td>
<td>0.4604</td>
<td>0.3833</td>
</tr>
<tr>
<td>transE + initialization</td>
<td>0.7135</td>
<td>0.5513</td>
</tr>
<tr>
<td>transE + initialization + regularization</td>
<td>0.7307</td>
<td>0.5580</td>
</tr>
<tr>
<td>transE + initialization + relation augmentation</td>
<td>0.7277</td>
<td>0.5746</td>
</tr>
<tr>
<td>transE + all enhancements</td>
<td>0.7396</td>
<td>0.5851</td>
</tr>
<tr>
<td>transD + all enhancements</td>
<td>0.7347</td>
<td>0.5851</td>
</tr>
<tr>
<td>transD + all enhancements</td>
<td>0.7347</td>
<td>0.5851</td>
</tr>
</tbody>
</table>

Table 2: Overall performances of different combinations of methods compared with the two baselines.

<table>
<thead>
<tr>
<th>Test examples of the relatedness tasks[7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word1</td>
</tr>
<tr>
<td>love</td>
</tr>
<tr>
<td>stock</td>
</tr>
<tr>
<td>money</td>
</tr>
<tr>
<td>lad</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test examples of the categorization tasks[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
</tr>
<tr>
<td>animal</td>
</tr>
<tr>
<td>creator</td>
</tr>
<tr>
<td>district</td>
</tr>
<tr>
<td>feeling</td>
</tr>
<tr>
<td>game</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test examples of the analogy tasks[13]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairs with the similar relationship</td>
</tr>
<tr>
<td>river:bed, flower:stem, hill:top, state:city</td>
</tr>
<tr>
<td>hill:top, car:wheels, mailbox:flag, bed:frame</td>
</tr>
<tr>
<td>lamp:shade, chair:legs, chair:back, barnyard:henhouse</td>
</tr>
<tr>
<td>river:delta, bed:frame, hill:top, house:roof</td>
</tr>
</tbody>
</table>

Table 3: Some examples of the relatedness tasks, categorization tasks and analogy tasks.

5.2 Visualize sense embeddings

One important advantage of our approach is that we can obtain intermediate sense embeddings for the same word. Table 7 shows the top-5 nearest neighbors for the word bank: the geographical and financial senses of bank. We can observe that our sense embeddings effectively capture distinctions between different senses of a word. It’s also worthy to demonstrate that our sense embedding group senses with close meaning but different POS-tagging together, e.g. bank_2 is close to bank_5 and bank_7, which is infeasible in Word2vec and GloVe.

6 CONCLUSION AND FUTURE WORKS

In this paper, we design a novel framework that combines both relation information and co-occurrence information in sense and word embeddings. Both regularization and initialization significantly improve the performances from using pure transE. In addition, we can easily incorporate additional relation information in either off-line or on-line manner. The experiment results show that our end-system outperforms the state-of-the-art approaches on the benchmark datasets, and the effective word embeddings can be used as a general components for further NLP tasks.

However, there are still some limitations and future works we plan to conduct. First, we only learn the embeddings for entities in WordNet. The major problem is that our framework learns from relations in WordNet and we cannot learn embeddings for those entities that are not part of WordNet. We can alleviate this problem by combining WordNet with additional entities from other knowledge bases (not only relations). Second, we miss the syntactic information. For example, given two words go and goes, they will have the same sense set in the WordNet so that both have the same word embedding. Currently, we only evaluate our word embeddings on those unsupervised tasks. We will apply them to the downstream tasks, such as word sense disambiguation or sentiment classification, and show their effectiveness.
Table 4: The comparison between two strategies to combine senses embeddings into word embeddings. Both transE and transD have GloVe initialization.

Table 5: Parameter Tuning for different regularization rate C with transE.

Table 6: TransD V.S. TransE w/wo relation augmentation. All the methods have GloVe initialization.

Table 7: Nearest neighbors to two senses bank₁ and bank₂ of the ambiguous noun "bank".
REFERENCES


