A Probabilistic Model for Learning Multi-Prototype Word Embeddings

Fei Tian[†], Hanjun Dai[*], Jiang Bian[‡], Bin Gao[‡], Rui Zhang[⋆], Enhong Chen[†], Tie-Yan Liu[‡]
[†]University of Science and Technology of China, Hefei, P.R.China
[*]Fudan University, Shanghai, P.R.China
[‡]Microsoft Research, Building 2, No. 5 Danling Street, Beijing, P.R.China
[⋆]Sun Yat-Sen University, Guangzhou, P.R.China

[†]tianfei@mail.ustc.edu.cn, [†]cheneh@ustc.edu.cn, [*]daihanjun@gmail.com,
[‡]{jibian, bingao, tyliu}@microsoft.com, [⋆]rayz0620@hotmail.com

Abstract

Distributed word representations have been widely used and proven to be useful in quite a few natural language processing and text mining tasks. Most of existing word embedding models aim at generating only one embedding vector for each individual word, which, however, limits their effectiveness because huge amounts of words are polysemous (such as *bank* and *star*). To address this problem, it is necessary to build multi embedding vectors to represent different meanings of a word respectively. Some recent studies attempted to train multi-prototype word embeddings through clustering context window features of the word. However, due to a large number of parameters to train, these methods yield limited scalability and are inefficient to be trained with big data. In this paper, we introduce a much more efficient method for learning multi embedding vectors for polysemous words. In particular, we first propose to model word polysemy from a probabilistic perspective and integrate it with the highly efficient continuous Skip-Gram model. Under this framework, we design an Expectation-Maximization algorithm to learn the word's multi embedding vectors. With much less parameters to train, our model can achieve comparable or even better results on word-similarity tasks compared with conventional methods.

1 Introduction

Distributed word representations usually refer to low dimensional and dense real value vectors (a.k.a. word embeddings) to represent words, which are assumed to convey semantic information contained in words. With the exploding text data on the Web and fast development of deep neural network technologies, distributed word embeddings have been effectively trained and widely used in a lot of text mining tasks (Bengio et al., 2003) (Morin and Bengio, 2005) (Mnih and Hinton, 2007) (Collobert et al., 2011) (Mikolov et al., 2010) (Mikolov et al., 2013b).

While word embedding plays an increasingly important role in many tasks, most of word embedding models, which assume one embedding vector for each individual word, suffer from a critical limitation for modeling tremendous polysemous words (e.g. *bank*, *left*, *doctor*). Using the same embedding vector to represent the different meanings (we will call *prototype* of a word in the rest of the paper) of a polysemous word is somehow unreasonable and sometimes it even hurts the model's expression ability.

To address this problem, some recent efforts, such as (Reisinger and Mooney, 2010) (Huang et al., 2012), have investigated how to obtain multi embedding vectors for the respective different prototypes of a polysemous word. Specifically, these works usually take a two-step approach: they first train single prototype word representations through a multi-layer neural network with the assumption that one word only yields single word embedding; then, they identify multi word embeddings for each polysemous word by clustering all its context window features, which are usually computed as the average of single prototype embeddings of its neighboring words in the context window.

Compared with traditional single prototype model, these models have demonstrated significant improvements in many semantic natural language processing (NLP) tasks. However, they suffer from a

crucial restriction in terms of scalability when facing exploding training text corpus, mainly due to the deep layers and huge amounts of parameters in the neural networks in these models. Moreover, the performance of these multi-prototype models is quite sensitive to the clustering algorithm and requires much effort in clustering implementation and parameter tuning. The lack of probabilistic explanation also refrains clustering based methods from being applied to many text mining tasks, such as language modeling.

To address these challenges, in this work, we propose a new probabilistic multi-prototype model and integrate it into a highly efficient continuous Skip-Gram model, which was recently introduced in the well-known Word2Vec toolkit (Mikolov et al., 2013b). Compared with conventional neural network language models which usually set up a multi-layer neural network, Word2Vec merely leverages a three-layer neural network to learn word embeddings, resulting in greatly decreased number of parameters and largely increased scalability. However, similar to most of existing word embedding models, Word2Vec also assumes *one embedding for one word*. We break this limitation by introducing a new probabilistic framework which employs hidden variables to indicate which prototype each word belongs to in the context. In this framework, the conditional probability of observing word w_O conditioned on the presence of neighboring word w_I (i.e. $P(w_O|w_I)$) can be formulated as a mixture model, where *mixtures* corresponds to w_I's different *prototypes*. This is a more natural way to define $P(w_O|w_I)$, since it has taken the polysemy of word w_I into consideration. After defining the model, we design an efficient Expectation-Maximization (EM) algorithm to learn various word embedding vectors corresponding to each of w_I's prototypes. Evaluations on widely used word similarity tasks demonstrate that our algorithm produces comparable or even better word embeddings compared with either clustering-based multi-prototype models or the original Skip-Gram model. Furthermore, as a unified way to obtain multi word embeddings, our proposed method can effectively avoid the sensitivity to the clustering algorithm applied by previous multi-prototype word embedding approach.

The following of the paper is organized as follows: we introduce related work in Section 2. Then, Section 3 describes our new model and algorithm in details and conducts a comparison in terms of complexity between our algorithm and the previous method. We present our experimental results in Section 4. The paper is concluded in Section 5.

2 Related Work

Since the initial work (Bengio et al., 2003), there have been quite a lot of neural network based models to obtain distributed word representations (Morin and Bengio, 2005) (Mnih and Hinton, 2007) (Mikolov et al., 2010) (Collobert et al., 2011) (Mikolov et al., 2013b). Most of these models assume that one word has only one embedding, except the work of Eric Huang (Huang et al., 2012), in which the authors propose to leverage global context information and multi-prototype embeddings to achieve performance gains in word similarity task. To obtain multi-prototype word embeddings, this work conducts clustering on a word's all context words' features in the corpus. The features are the embedding vectors trained previously via a three-layer neural network. Each cluster's centroid is regarded as the embedding vector for each prototype. Their reported experimental results verify the importance of considering multi-prototype models.

Note that (Reisinger and Mooney, 2010) also proposes to deal with the word polysemy problem by assigning to each prototype a real value vector. However their embedding vectors are obtained through a tf-idf counting model, which is usually called as distributional representations (Turian et al., 2010), rather than through a neural network. Therefore, we do not regard their paper as very related to our work. The similar statement holds for other works on vector model for word meaning in context such as (Erk and Padó, 2008) (Thater et al., 2011) (Reddy et al., 2011) (Van de Cruys et al., 2011).

Our model is mainly based on the recent proposed Word2Vec model, more concretely, the continuous Skip-Gram model (Mikolov et al., 2013a) (Mikolov et al., 2013b). The continuous Skip-Gram model specifies the probability of observing the context words conditioned on the central word w_I in the window via a three-layer neural network. With less parameters to train (thus higher scalability), Word2Vec discovers interesting analogical semantic relations between words like *Japan - Tokyo = France - Paris*.

3 Model Description

In this section, we introduce our algorithm for learning multi-prototype embeddings in details. In particular, since our new model is based on the continuous Skip-Gram model, we first make a brief introduction to the Skip-Gram model. Then, we present our new multi-prototype algorithm and how we integrate it into the Skip-Gram model. After that, we propose an EM algorithm to conduct the training process. We also conduct a comparison on the number of parameters between the new EM algorithm and the state-of-the-art multi-prototype model proposed in (Huang et al., 2012), which can illustrate the efficiency superior of our algorithm.

3.1 Multi-Prototype Skip-Gram Model

In contrast to the conventional ways of using context words to predict the next word or the central word, the Skip-Gram model (Mikolov et al., 2013b) aims to leverage the central word to predict its context words. Specifically, assuming that the central word is w_I and one of its neighboring word is w_O, $P(w_O|w_I)$ is modeled in the following way:

$$P(w_O|w_I) = \frac{exp(V_{w_I}^T U_{w_O})}{\sum_{w \in W} exp(V_{w_I}^T U_w)}, \tag{1}$$

where W denotes the dictionary consisting of all words, $U_w \in R^d$ and $V_w \in R^d$ represent the d-dimensional 'output' and 'input' embedding vectors of word w, respectively. Note that all the parameters to be learned are the input and output embedding vectors of all words, i.e. $U = \{U_w | w \in W\}$ and $V = \{V_w | w \in W\}$. This corresponds to a three-layer neural network, in which U and V denote the two parameter matrices of the neural network. Compared with the conventional neural networks employed in the literature which yield at least four layers (including the look-up table layer), the Skip-Gram model greatly reduces the number of parameters and thus gives rise to a significant improvement in terms of training efficiency.

Our proposed Multi-Prototype Skip-Gram model is similar to the original Skip-Gram model in that it also aims to model $P(w_O|w_I)$ and uses two matrices (the input and output embedding matrices) as the parameters. The difference lies in that given word w_I, the occurrence of word w_O is described as a finite mixture model, in which each mixture corresponds to a prototype of word w_I. To be specific, suppose that word w has N_w prototypes and it appears in its h_w-th prototype, i.e., $h_w \in \{1, \cdots, N_w\}$ is the index of w's prototype. Then $P(w_O|w_I)$ is expanded as:

$$p(w_O|w_I) = \sum_{i=1}^{N_{w_I}} P(w_O|h_{w_I} = i, w_I) P(h_{w_I} = i|w_I) \tag{2}$$

$$= \sum_{i=1}^{N_{w_I}} \frac{exp(U_{w_O}^T V_{w_I,i})}{\sum_{w \in W} exp(U_w^T V_{w_I,i})} P(h_{w_I} = i|w_I), \tag{3}$$

where $V_{w_I,i} \in R^d$ refers to the embedding vector of w_I's i-th prototype. This equation states that $P(w_O|w_I)$ is a weighted average of the probabilities of observing w_O conditioned on the appearance of w_I's every prototype. The probability $P(w_O|h_{w_I} = i, w_I)$ takes the similar softmax form to equation (1) and the weight is specified as a prior probability of word w_I falls in its every prototype.

The general idea behind the Multi-Prototype Skip-Gram model is very intuitive: the surrounding words under different prototypes of the same word are usually different. For example, when the word *bank* refers to the side of a river, it is very possible to observe the corresponding context words such as *river*, *water*, and *slope*; however, when *bank* falls into the meaning of the financial organization, the surrounding word set is likely to be comprised of quite different words, such as *money*, *account*, and *investment*.

The probability formulation in (3) brings much computation cost because of the linear dependency of $|W|$ in the denominator $\sum_{w \in W} exp(U_w^T V_{w_I,i})$. To address this issue, several efficient methods have been proposed such as Hierarchical Softmax Tree (Morin and Bengio, 2005) (Mnih and Kavukcuoglu, 2013) and Negative Sampling (Mnih and Kavukcuoglu, 2013) (Mikolov et al., 2013b). Taking Hierarchical

Softmax Tree as an example, through a binary tree in which every word is a leaf node, word w_O is associated with a binary vector $b^{(w_O)} \in \{-1,+1\}^{L_{w_O}}$ specifying a path from the root of the tree to leaf w_O, where L_{w_O} is the length of vector $b^{(w_O)}$. Then the conditional probability is described as

$$P(w_O|h_{w_I}=i,w_I)=\prod_{t=1}^{L_{w_O}}P\left(b_t^{(w_O)}|w_I,h_{w_I}=i\right)=\prod_{t=1}^{L_{w_O}}\varsigma\left(b_t^{(w_O)}U_{w_O,t}^T V_{w_I,i}\right), \tag{4}$$

where $\varsigma(x)=1/(1+exp(-x))$ is the sigmoid function, and $U_{w_O,t}$ specifies the d-dimensional parameter vector associated with the t-th node in the path from the root to the leaf node w_O. Substituting (4) into (2) to replace the large softmax operator in (3) leads to a much more efficient probability form.

3.2 EM Algorithm

In this section, we describe the EM algorithm adopted to train the Multi-Prototype Skip-Gram model. Without loss of generality, we will focus on obtaining multi embeddings for a specified word $w \in W$ with N_w prototypes. Word w's embedding vectors are denoted as $V_w \in R^{d\times N_w}$. Suppose there are M word pairs for training: $\{(w_1,w),(w_2,w),\cdots,(w_M,w)\}$, where all the inputs words (i.e., word w) are the same, and the set of output words to be predicted are denoted as $\mathbb{X}=\{w_1,w_2,\cdots,w_M\}$. That is, $\mathbb{X}$ are M surrounding words of w in the training corpus.

For ease of reference and without loss of generality, we make some changes to the notations in Section 3.1. We will use h_m as the index of w's prototype in the pair (w_m,w), $m\in\{1,2,\cdots,M\}$. Besides, some new notations are introduced: $P(h_w=i|w_I)$ is simplified as π_i, and $\gamma_{m,k}$, where $m\in\{1,2,\cdots,M\}$, $k\in\{1,2,\cdots N_w\}$, are the hidden binary variables indicating whether the m-th presence of word w is in its k-th prototype, i.e. $\gamma_{m,k}=1_{h_m=k}$, where 1 is the indicator function. Other notations are the same as before: $V_{w,i}\in R^d$ is the embedding vector for word w's i-th prototype, $U_{w,t}\in R^d$ is the embedding vector for the t-th node on the path from the tree root to the leaf node representing word w, and $b_t^{(w)}\in\{-1,1\}$ is the t-th bit of the binary coding vector of word w along its corresponding path on the Hierarchical Softmax Tree.

Then the parameter set we aim to learn is $\Theta=\{\pi_1,\cdots,\pi_{N_w};U;V_w\}$. The hidden variable set is $\Gamma=\{\gamma_{m,k}|m\in(1,2,\cdots,M),k\in(1,2,\cdots,N_w)\}$. Considering equation (2) and (4), we have the log likelihood of $\mathbb{X}$ as below:

$$\begin{aligned}\log P(\mathbb{X},\Gamma|\Theta)&=\sum_{m=1}^{M}\sum_{k=1}^{N_w}\gamma_{m,k}\big(\log\pi_k+\log P(w_m|h_m=k,w)\big)\\&=\sum_{m=1}^{M}\sum_{k=1}^{N_w}\gamma_{m,k}\big(\log\pi_k+\sum_{t=1}^{L_{w_m}}\log\varsigma(b_t^{(w_m)}U_{w_m,t}^T V_{w,k})\big).\end{aligned} \tag{5}$$

With equation (5), the E-Step and M-Step are:

E-Step:

The conditional expectation of hidden variable $\gamma_{m,k}$, denoted as $\hat{\gamma}_{m,k}$, is:

$$\hat{\gamma}_{m,k}=P(\gamma_{m,k}=1|\mathbb{X},\Theta)=\frac{\pi_k P(w_m|h_m=k,w)}{\sum_{i=1}^{N_w}\pi_i P(w_m|h_m=i,w)}. \tag{6}$$

The Q function w.r.t. the parameters at the i-th iteration $\theta^{(i)}$ is written as:

$$\begin{aligned}Q(\theta,\theta^{(i)})&=\sum_{k=1}^{N_w}\sum_{m=1}^{M}\hat{\gamma}_{m,k}\left(\log\pi_k+\log P(w_m|h_m=k,w)\right)\\&=\sum_{m=1}^{M}\sum_{k=1}^{N_w}\hat{\gamma}_{m,k}\big(\log\pi_k+\sum_{t=1}^{L_{w_m}}\log\varsigma(b_t^{(w_m)}U_{w_m,t}^T V_{w,k})\big).\end{aligned} \tag{7}$$

M-Step:

π can be updated by

$$\pi_k = \frac{\sum_{m=1}^{M} \hat{\gamma}_{m,k}}{M}, \quad k = 1, 2 \cdots, N_w. \tag{8}$$

We leave the detailed derivations for equation (6), (7), and (8) to the appendix of the paper. Then we discuss how we obtain the update of the embedding parameters $U_{w_m,t}$ and $V_{w,k}$. Note that the optimization problem is non-convex, and it is hard to compute the exact solution of $\frac{\partial Q}{\partial U_{w_m,t}} = 0$ and $\frac{\partial Q}{\partial V_{w,k}} = 0$. Therefore, we use gradient ascent to optimize in the M-step. The gradients of Q function w.r.t. embedding vectors are given by:

$$\frac{\partial Q}{\partial U_{w_m,t}} = \sum_{k=1}^{N_w} \hat{\gamma}_{m,k} b_t^{(w_m)} \big(1 - \varsigma(b_t^{(w_m)} U_{w_m,t}^T V_{w,k})\big) V_{w,k}, \tag{9}$$

$$\frac{\partial Q}{\partial V_{w,k}} = \sum_{m=1}^{M} \hat{\gamma}_{m,k} \sum_{t=1}^{L_{w_m}} b_t^{(w_m)} \big(1 - \varsigma(b_t^{(w_m)} U_{w_m,t}^T V_{w,k})\big) U_{w_m,t}. \tag{10}$$

Iterating between **E-Step** and **M-Step** till the convergence of the value of function Q makes the EM algorithm complete.

In order to enhance the scalability of our approach, we propose a fast computing method to boost the implementation of the EM algorithm. Note that the most expensive computing operations in both the E-Step and M-Step are the inner product of the input and output embedding vectors, as well as the sigmoid function. However, if we take the Hierarchical Softmax Tree form as shown in Equation (4) to model $P(w_m|h_m = i, w)$, and perform only one step gradient ascent in M-Step, the aforementioned two expensive operations in M-Step will be avoided by leveraging the pre-computed results in the E-Step. Specifically, since the gradient of the function $f(x) = \log \varsigma(x)$ is given by $f'(x) = 1 - \varsigma(x)$, the sigmoid values computed in the E-Step to obtain $P(w_m|h_m = i, w)$ (i.e. the term $\varsigma(b_t^{(w_m)} U_{w_m,t}^T V_{w,k})$ in equation (5), (9), and (10)) can be re-used to derive the gradients in the M-Step.

However, such enhanced computation method cannot benefit the second order optimization methods in the M-Step such as L-BFGS and Conjugate Gradient, since they usually rely on multiple iterations to converge. In fact, we tried these two optimization methods in our experiments but they have brought no improvement compared with simple one-step gradient ascent method.

3.3 Model Comparison

To show that our model is more scalable than the former multi-prototype model in (Huang et al., 2012) (We denote it as EHModel in the rest of the paper), we conduct a comparison on the number of parameters with respect to each of these two models in this subsection.

We use $n_{embedding}$ and n_{window} to denote the numbers of all word embedding vectors and context window words, respectively. It is clear that $n_{embeddings} = \sum_{w \in W} N_w$. EHModel aims to compute two scores, i.e., the local score and the global score, both with hidden layer node activations. We denote the hidden layer node number as h_l and h_g for these two scores. The parameter numbers are listed in Table 1.

Model	EHModel	Our Model
#parameters	$dn_{words} + dn_{embeddings} + (dn_{window} + 1)h_l + (2d + 1)h_g$	$dn_{words} + dn_{embeddings}$

Table 1: Comparison of parameter numbers of two models

Note that d in Table 1 denotes the embedding vector size. It can be observed that EHModel has $(dn_{window} + 1)h_l + (2d + 1)h_g$ more parameters than our model, which is mainly because EHModel has one more layer in the neural network and it considers global context. In previous study (Huang et al., 2012), d, n_{window}, h_l, and h_g are set to be 50, 10, 100, 100, respectively, which greatly increases the gap of parameter numbers between the two models.

4 Experiments

In this section, we will present our experimental settings and results. Particularly, we first describe the data collection and the training configuration we used in the experiments; then, we conduct a qualitative case study followed by quantitative evaluation results on a public word similarity task to demonstrate the performance of our proposed model.

4.1 Experimental Setup

Dataset: To make a fair comparison with the state-of-the-art methods, we employ a publicly available dataset, which is used in (Huang et al., 2012), to train word embeddings in our experiments. Particularly, this training corpus is a snapshot of Wikipedia at April, 2010 (Shaoul, 2010), which contains about 990 million tokens. We removed the infrequent words from this corpus and kept a dictionary of about 1 million most frequent words. Similar to Word2Vec, we removed pure digit words such as *2014* as well as about 100 stop words like *how*, *for*, and *we*.

Training Configuration: In order to boost the training speed, we take advantage of the Hierarchical Softmax Tree structure. More concretely, we use the Huffman tree structure, as introduced in Word2Vec, to further increase the training speed. All the embedding size, including both word embedding vectors and the Huffman tree node embedding vectors, are set to be 50, which is the same as the size used in (Huang et al., 2012). To train word embedding, we set the context window size as 10, i.e., for a word w, 10 of the closest neighboring words to w are regarded as ws contexts. For the numbers of word prototypes, i.e., N_w introduced in Section 3.2, we set the top 7 thousand frequent words as multi-prototype words by experience, with all of them having 10 prototypes (i.e. $N_w = 10$).

During the training process, we used the same strategy to set the learning rate as what Word2Vec did. Specifically, we set the initial learning rate to 0.025 and diminished the value linearly along with the increasing number of training words. Our experimental results illustrate that this learning rate strategy can lead to the best results for our algorithm.

For the hyper parameters of the EM algorithm, we set the batch size to 1, i.e. $M = 1$ in Section 3.2, since our experimental results reveal that smaller batch size can result in better experimental results. The reason is explained as the following. Our optimization problem is highly non-convex. Smaller batch size yields more frequent updates of parameters, and thus avoids trapping in local optima, while larger batch size, associated with more infrequent parameter updating, may cause higher probability to encounter local optima. In our experiments, we observe that only one iteration of E-Step and M-Step can reach the embedding vectors with good enough performance on the word similarity task, whereas increasing the iteration number just leads to slight performance improvement with much longer training time. Under the above configuration, our model runs about three times faster than EHModel.

4.2 Case Study

This section gives some qualitative evaluations of our model by demonstrating how our model can effectively identify multi-prototype word embeddings on some specific cases. In Table 2, we list several polysemous words. For each word, we pick some of their prototypes learned by our model, including the prototype prior probability (i.e. π_i introduced in Section 3.2) and three of the most similar words with each prototype, respectively. The similarity is calculated by the cosine similarity score between the embedding vectors.

From the table we can observe some interesting results of the multi-prototype embedding vectors produced by our model:

- For a polysemous word, its different embedding vectors represent its different semantic meanings. For example, the first embedding vector of the word *apple* corresponds to its sense as a kind of fruit, whereas the second one represents its meaning as an IT company.

- The prior probability reflects the likelihood of the occurrence of various prototypes to some extent. For example, the word *cell* is more likely to represent the meaning of the smallest part of living structure (with probability 0.81), than to be used as the meaning of *cellphone* (with probability

Word	Prior Probability	Most Similar Words
apple_1	0.82	strawberry, cherry, blueberry
apple_2	0.17	iphone, macintosh, microsoft
bank_1	0.15	river, canal, waterway
bank_2	0.6	citibank, jpmorgan, bancorp
bank_3	0.25	stock, exchange, banking
cell_1	0.09	phones, cellphones, mobile
cell_2	0.81	protein, tissues, lysis
cell_3	0.01	locked, escape, handcuffed

Table 2: Most similar words with different prototypes of the same word

0.09) or *prisoned* (with probability 0.01). Note that the three prior probability scores of *cell* do not sum to 1. The reason is that there are some other embeddings not presented in the table which are found to have high similarities with the three embeddings. We do not present them due to the space limitation.

- By setting the prototype number to a fairly large value (e.g. $N_w = 10$), the model tends to learn more fine-grained separations of the word's different meanings. For example, we can observe from Table 2 that the second and the third prototypes of the word *bank* seem similar to each other as both of them denote a financial concept. However, there are subtle differences between them: the second prototype represents concrete banks, such as citibank and jpmorgan, whereas the third one denotes what is done in the banks, since it is most similar to the words *stock*, *exchange*, and *banking*. We believe that such a fine-grained separation will bring more expressiveness to the multi-prototype word embeddings learned by our model.

4.3 Results on Word Similarity in Context Dataset

In this subsection, we give quantitative comparison of our method with conventional word embedding models, including Word2Vec and EHModel (Huang et al., 2012).

The task we perform is the word similarity evaluation introduced in (Huang et al., 2012). Word similarity tasks evaluate a model's performance by calculating the Spearman's rank correlation between the ranking of ground truth similarity scores (given by human labeling) and the ranking based on the similarity scores produced by the model. Traditional word similarity tasks such as WordSim353 (Finkelstein et al., 2001) and RG (Rubenstein and Goodenough, 1965) are not suitable for evaluating multi-prototype models since there is neither enough number of polysemous words in these datasets nor context information to infer the prototype index. To address this issue, a new word similarity benchmark dataset including context information was released in (Huang et al., 2012). Following (Luong et al., 2013), we use SCWS to denote this dataset. Similar to WordSim353, SCWS contains some word pairs (concretely, 2003 pairs), together with human labeled similarity scores for these word pairs. What makes SCWS different from WS353 is that the words in SCWS are contained in sentences, i.e., there are 2003 pairs of sentences containing these words, while words in WS353 are not associated with sentences. Therefore, the human labeled scores are based on the meanings of the words in the context. Given the presence of the context, the word similarity scores, especially those scores depending on polysemous words, are much more convincing for evaluating different models' performance in our experiments.

Then, we propose a method to compute the similarity score for a pair of words $\{w_1, w_2\}$ in the context based on our model. Suppose that the context of a word w is defined as all its neighboring words in a $T+1$ sized window, where w is the central word in the window. We use $Context_1 = \{c_1^1, c_2^1, \cdots, c_T^1\}$ and $Context_2 = \{c_1^2, c_2^2, \cdots, c_T^2\}$ to separately denote the context of w_1 and w_2, where c_t^1 and c_t^2 are the t-th context word of w_1 and w_2, respectively. According to Bayesian rule, we have that for $i \in \{1, 2, \cdots, N_{w_1}\}$:

$$\begin{aligned} P(h_{w_1} = i | Context_1, w_1) &\propto P(Context_1 | h_{w_1} = i, w_1) P(h_{w_1} = i | w_1) \\ &= \prod_{t=1}^{T} P(c_t^1 | h_{w_1} = i, w_1) P(h_{w_1} = i | w_1), \end{aligned} \tag{11}$$

where $P(c_t^1|h_{w_1}=i,w_1)$ can be calculated by equation (4) and $P(h_{w_1}=i|w_1)$ is the prior probability we learned in the EM algorithm (equation (8)). The similar equation holds for word w_2 as well. Here we make an assumption that the context words are independent with each other given the central word. Furthermore, suppose that the most likely prototype index for w_1 given $Context_1$ is $\hat{h}_{w_1}$, i.e., we denote $\hat{h}_{w_1} = \arg\max_{i\in\{1,2,\cdots,N_{w_1}\}} P(h_{w_1}=i|Context_1,w_1)$. Similarly, $\hat{h}_{w_2}$ is denoted as the corresponding meaning for w_2.

We calculate two similarity scores base on equation (11), i.e., MaxSim Score and WeightedSim Score:

$$MaxSim(w_1,w_2) = Cosine(V_{w_1,\hat{h}_{w_1}}, V_{w_2,\hat{h}_{w_2}}), \tag{12}$$

$$WeightedSim(w_1,w_2) = \sum_{i=1}^{N_{w_1}}\sum_{j=1}^{N_{w_2}} P(h_{w_1}=i|Context_1,w_1)P(h_{w_2}=j|Context_2,w_2)Cosine(V_{w_1,i},V_{w_2,j}). \tag{13}$$

In the above similarity scores, $Cosine(x,y)$ denotes the cosine similarity score of vector x and y, and $V_{w,i} \in R^d$ is the embedding vector for the word w's i-th prototype.

The detailed experimental results are listed in Table 3, where ρ refers to the Spearman's rank correlation. The higher value of ρ indicates the better performance. The performance score of EHModel is borrowed from its original paper (Huang et al., 2012). For Word2Vec model, we use Hierarchical Huffman Tree rather than Negative Sampling to do the acceleration. Our **Model_M** uses the MaxSim score in testing and our **Model_W** uses the WeightedSim score. All of these models are run on the same aforementioned Wikipedia corpus, with the dimension of the embedding space to be 50.

From the table, we can observe that our **Model_W** (65.4%) outperforms the original Word2Vec model (61.7%), and achieves almost the same performance with the state-of-the-art EHModel (65.7%). Among the two similarity measures used in testing, the WeightedSim score performs better (65.4%) than the MaxSim score (63.6%), indicating that the overall consideration of all prototype probabilities are more effective.

Model	$\rho \times 100$
Word2Vec	61.7
EHModel	65.7
Model_M	63.6
Model_W	65.4

Table 3: Spearman's rank correlations on SCWS dataset.

5 Conclusion

In this paper, we introduce a fast and probabilistic method to generate multiple embedding vectors for polysemous words, based on the continuous Skip-Gram model. On one hand, our method addresses the drawbacks of the original Word2Vec model by leveraging multi-prototype word embeddings; on the other hand, our model yields much less complexity without performance loss compared with the former clustering based multi-prototype algorithms. In addition, the probabilistic framework of our method avoids the extra efforts to perform clustering besides training word embeddings.

For the future work, we plan to apply the proposed probabilistic framework to other neural network language models. Moreover, we would like to apply the multi-prototype embeddings to more real world text mining tasks, such as information retrieval and knowledge mining, with the expectation that the multi-prototype embeddings produced by our model will benefit these tasks.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. In *Journal of Machine Learning Research*, pages 1137–1155.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Katrin Erk and Sebastian Padó. 2008. A structured vector space model for word meaning in context. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 897–906, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics.

Minh-Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. *CoNLL-2013*, 104.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2265–2273.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252.

Siva Reddy, Ioannis P Klapaftis, Diana McCarthy, and Suresh Manandhar. 2011. Dynamic and static prototype vectors for semantic composition. In *IJCNLP*, pages 705–713.

Joseph Reisinger and Raymond J Mooney. 2010. Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 109–117. Association for Computational Linguistics.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Westbury C Shaoul, C. 2010. The westbury lab wikipedia corpus.

Stefan Thater, Hagen Fürstenau, and Manfred Pinkal. 2011. Word meaning in context: A simple and effective vector model. In *IJCNLP*, pages 1134–1143.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics.

Tim Van de Cruys, Thierry Poibeau, and Anna Korhonen. 2011. Latent vector weighting for word meaning in context. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1012–1022, Stroudsburg, PA, USA. Association for Computational Linguistics.

6 Appendix

6.1 Derivations for the EM Algorithm

We give detailed derivations for the updating rules used in the EM algorithms in Section 3.2., i.e., the derivations for equation (6), (7), and (8).

According to the properties of conditional probability, we have

$$\begin{aligned}\hat{\gamma}_{m,k} = P(\gamma_{m,k}=1|\mathbb{X},\Theta) &= \frac{P(\gamma_{m,k}=1,\mathbb{X}|\Theta)}{\sum_{i=1}^{N_w} P(\gamma_{m,i}=1,\mathbb{X}|\Theta)} \\ &= \frac{P(\gamma_{m,k}=1|\Theta)P(\mathbb{X}|\gamma_{m,k}=1,\Theta)}{\sum_{i=1}^{N_w} P(\gamma_{m,i}=1|\Theta)P(\mathbb{X}|\gamma_{m,i}=1,\Theta)} \\ &= \frac{\pi_k P(w_m|h_m=k,w)}{\sum_{i=1}^{N_w} \pi_i P(w_m|h_m=i,w)}.\end{aligned} \tag{14}$$

From equation (7), the Q function is calculated as:

$$\begin{aligned}Q(\theta,\theta^{(i)}) &= E[\log P(\mathbb{X},\Gamma|\Theta)|\Theta^{(i)}] \\ &= \sum_{k=1}^{N_w}\sum_{m=1}^{M} E[\gamma_{m,k}|\Theta^{(i)}]\big(\log\pi_k + \sum_{t=1}^{L_{w_m}} \log\varsigma(b_t^{(w_m)} U_{w_m,t}^T V_{w,k})\big) \\ &= \sum_{k=1}^{N_w}\sum_{m=1}^{M} \hat{\gamma}_{m,k}\big(\log\pi_k + \sum_{t=1}^{L_{w_m}} \log\varsigma(b_t^{(w_m)} U_{w_m,t}^T V_{w,k})\big) \\ &= \sum_{m=1}^{M}\sum_{k=1}^{N_w} \hat{\gamma}_{m,k}\big(\log\pi_k + \sum_{t=1}^{L_{w_m}} \log\varsigma(b_t^{(w_m)} U_{w_m,t}^T V_{w,k})\big).\end{aligned} \tag{15}$$

Then we give the derivations for π's updating rule, i.e., equation (8). Note that for parameters π_k, $k=\{1,2,\cdots,N_w\}$, they need to satisfy the condition that $\sum_{k=1}^{N_w}\pi_k = 1$. From equation (7) (or equivalently equation (15)), the loss with regard to π is:

$$L_{[\pi]} = \sum_{m=1}^{M}\sum_{k=1}^{N_w} \hat{\gamma}_{m,k}\log\pi_k + \lambda\big(\sum_{k=1}^{N_w}\pi_k - 1\big), \tag{16}$$

where λ is the Language multiplier. Letting $\frac{\partial L_{[\pi]}}{\partial\pi} = 0$, we obtain:

$$\pi_k \propto \sum_{m=1}^{M} \hat{\gamma}_{m,k}. \tag{17}$$

Further considering the fact that $\sum_{k=1}^{N_w}\sum_{m=1}^{M}\hat{\gamma}_{m,k} = M$, we have $\pi_k = \frac{\sum_{m=1}^{M}\hat{\gamma}_{m,k}}{M}$.