Unsupervised Learning of Word Semantic Embedding using the Deep Structured Semantic Model

Xinying Song, Xiaodong He, Jianfeng Gao, Li Deng

Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.

{xinson,xiaohe,jfgao,deng}@microsoft.com

Technical Report

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Abstract

Deep neural network (DNN) based natural language processing models rely on a word embedding matrix to transform raw words into vectors. Recently, a deep structured semantic model (DSSM) has been proposed to project raw text to a continuously-valued vector for Web Search. In this technical report, we propose learning word embedding using DSSM. We show that the DSSM trained on large body of text can produce meaningful word embedding vectors as demonstrated on semantic word clustering and semantic word analogy tasks.

1 Introduction

Deep neural network (DNN) based semantic models such as semantic hashing (SH) have been proposed for information retrieval [2][3]. For example, Salakhutdinov and Hinton extended the semantic modeling using deep auto-encoders [2][3]. They demonstrated that the hierarchical semantic structure embedded in the query and the document can be extracted via deep learning. Superior performance to the conventional method of Latent Semantic Analysis (LSA) was demonstrated in [2][3]. In semantic hashing and other Deep neural network (DNN) based natural language processing models, usually the first layer of the network serves the purpose of converting the word into a low-dimension vector representation, as known as word embedding, which is either learned separately on large body of raw text, or learned jointly with other parameters of the network [1][2][3]. For example, a feedforward neural network was used to jointly learn the word embedding and a statistical language model [14][15][16][17]; recurrent neural language models was also proposed to handle arbitrary long contexts and more complex patterns [18]; Mnih and Kavukcuoglu [13] proposed a feedforward neural network to maximize the similarity between the embedding vectors of the word and of the context. Recently, a Deep Structured Semantic Models (DSSM) for Web search was proposed in [6], which is reported to outperform significantly semantic hashing and other conventional semantic models.

In this study, extending from the research discussed above, we developed a Deep Structured Semantic Model (DSSM) to learn word embedding. We show that the DSSM trained on large body of text can produce meaningful word embedding vectors as demonstrated on semantic word clustering and semantic word analogy tasks.

2 Deep Structured Semantic Model

2.1 The Overall Architecture of the Deep Structured Semantic Model

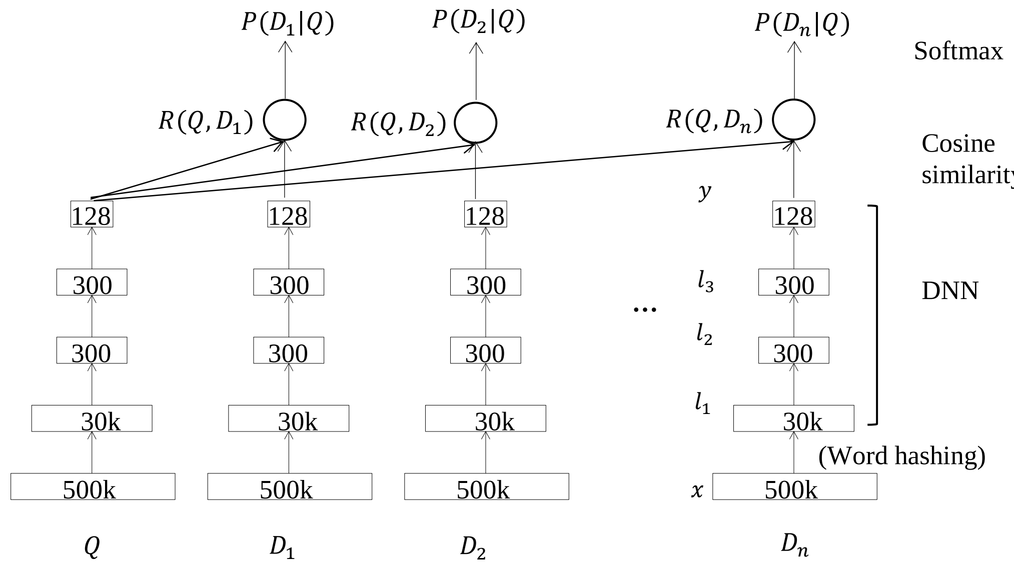

Figure 1: The illustration of the overall structure of the DSSM.

The overall architecture of the proposed deep structured semantic model (DSSM) is shown in Figure 1. DSSM was originally proposed with Web search as the example task that predicts relevant documents for a given query. We use Web search to introduce DSSM in this section and discuss how to apply DSSM to learn word embeddings in later sections.

The high-dimensional input to the DNN is a term vector, e.g., raw counts of terms in a query or a document without normalization (a.k.a. bag-of-words), and the output of the DNN is a concept vector in a low-dimensional semantic feature space. Given this DNN, queries and documents can be mapped to their corresponding semantic concept vectors. Then, another layer on top of the outputs of DNNs is applied, where at each node is the cosine similarity score between the concept vectors of the query and each of the documents, respectively. Then a softmax smoothing is applied to convert the cosine similarity scores into conditional likelihood.

Let's denote by x as the input term vector, y as the output vector, $l_i, i = 1, \dots, N-1$, as the intermediate hidden layers, W_i as the i-th weight matrix, and b_i as the i-th bias term, we have

$$\begin{aligned} l_1 &= W_1 x \\ l_i &= f(W_i l_{i-1} + b_i), i = 2, \dots, N-1 \\ y &= f(W_N l_{N-1} + b_N) \end{aligned} \quad (1)$$

where we use the $tanh$ as the activation function at the output layer and the hidden layers $l_i, i = 2, \dots, N-1$:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (2)$$

The semantic relevance score between a query Q and a document D is then measured as:

$$R(Q,D) = \text{cosine}(y_Q, y_D) = \frac{{y_Q}^T y_D}{\|y_Q\| \|y_D\|} \quad (3)$$

where y_Q and y_D are the concept vectors of the query and the document, respectively. In Web search, given the query, the documents are sorted by their semantic relevance scores.

Conventionally, the size of the term vector, which can be viewed as the raw bag-of-words features in IR, is equal to that of the vocabulary that is used for indexing the Web document collection. The vocabulary size is usually very large in real-world Web search tasks. Therefore, when using term vector as the input, the size of the input layer of the neural network would be unmanageable for inference and model training. To address this problem, we have developed two methods of "word hashing" for the first layer of the DNN, as indicated in the lower portion of Figure 1. This layer consists of only linear hidden units in which the weight matrix of a very large size is not learned. In the following section, we will describe the word hashing method in detail.

2.2 Word Hashing

The two word hashing methods described here aim to reduce the dimensionality of the bag-of-words term vectors without requiring learning. The first method is based on the well-known random projection. The second is based on letter-n-grams, and has an additional advantage of making the generalization ability from the training set to the test set better. That is, the unseen words in the test set are likely to have their letter-n-gram presentations present in the training set.

2.2.1 Random Projection based Word Hashing

The method of sparse random projection [8] is originally proposed to perform dimensionality reduction. We apply it to word hashing.

The sparse random projection technique projects each high-dimensional input vector (e.g., 500k) into a low-dimensional space (a few thousands) using a sparse matrix R whose entries are sampled i.i.d. from a distribution over {0, 1, -1}. Entries of 1 and -1 are with an equal probability, and $P(R_{ij} = 0) = 1 - \frac{1}{\sqrt{d}}$, where d is the original input dimension.

Since the projection matrix is sampled without training, the computational cost is negligible for even very large vocabularies. However, this method cannot deal with the mapping of those terms that are unseen in the training set.

2.2.2 Letter-n-gram based Word Hashing

A *word hashing* method is also developed in this work based on the letter-n-gram count distribution for our task. Given a word (e.g. *good*), we first add word starting and ending marks to the word (e.g. *#good#*). Then, we break the word into letter-n-grams (e.g. letter trigrams: *#go*, *goo*, *ood*, *od#*). Finally, the word is represented using a vector of letter-n-grams.

One problem of this method is *collision*, i.e., two different words could have the same letter-n-gram vector representation. Table 1 shows some statistics of word hashing on two vocabularies. Compared with the original size of the one-hot vector, word hashing allows us to represent a query or a document using a vector with much lower dimensionality. Take the 40K-word vocabulary as an example. Each word can be represented by a 10,306-dimentional vector using letter trigrams, giving a four-fold dimensionality reduction with few collisions. The reduction of dimensionality is even more significant when the technique is applied to a larger vocabulary. As shown in Table 2, each word in the 500K-word vocabulary can be represented by a 30,621 dimensional vector using letter trigrams, a reduction of 16-fold in dimensionality with a negligible collision rate of 0.0044% (22/500,000).

	Letter-Bigram		**Letter-Trigram**	
Word SizeSize	**Token Size**	**Collision**	**Token Size**	**Collision**
40k	1107	18	10306	2
500k	1607	1192	30621	22

Table 1: Word hashing token size and collision numbers as a function of the vocabulary size and the type of letter bigrams and trigrams.

While the number of English words can be unlimited, the number of letter-n-grams in English (or other similar languages) is often limited. Moreover, word hashing is able to map the morphological variations of the same word to the points that are close to each other in the letter-n-gram space. More importantly, while a word unseen in the training set always cause difficulties in word-based representations, it is not the case where the letter-n-gram based representation is used. The only risk is the minor representation collision as quantified in Table 2. Thus, letter-n-gram based word hashing is robust to the out-of-vocabulary problem, allowing us to scale up the DNN solution to the Web search tasks where extremely large vocabularies are desirable. We will demonstrate the benefit of the technique empirically in Section 4.

The letter-n-gram based word hashing method uses a fixed (i.e., non-adaptive) linear transformation matrix, through which a term vector in the input layer is projected to a letter-n-gram vector in the next layer higher up, as shown in Figure 1. Since the letter-n-gram vector is of a much lower dimensionality, DNN learning can be carried out effectively.

2.3 Learning

The clickthrough logs consist of a list of queries and their clicked documents. We assume that a query is relevant to the documents that are clicked on for that query. Therefore, the DSN can be effectively learned by maximizing the conditional likelihood of the clicked documents given the queries.

First, as illustrated in Figure 1, we compute the probability of a document given a query from the semantic relevance score between them through a softmax function

$$P(D|Q) = \frac{\exp\big(\gamma R(Q,D)\big)}{\sum_{D' \in \boldsymbol{D}} \exp\big(\gamma R(Q,D')\big)} \qquad (4)$$

where γ is a smoothing factor in the softmax function, which is set empirically on a held-out data set in our experiment. $\boldsymbol{D}$ denotes the set of candidate documents to be ranked. Ideally, $\boldsymbol{D}$ should contain all possible documents. In practice, for each (query, clicked-document) pair, denoted by (Q, D^+) where Q is a query and D^+ is the clicked document, we approximate $\boldsymbol{D}$ by including D^+ and four randomly selected unclicked documents, denote by $\{D_j^-; j = 1, \dots, 4\}$.

In training, the model parameters are estimated to maximize the likelihood of the clicked documents given the queries across the training set. Equivalently, we need to minimize the following loss function

$$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q) \qquad (5)$$

where Λ denotes the parameter set of the neural networks $\{W_i, b_i\}$.

Since $L(\Lambda)$ is differentiable w.r.t. to Λ. The model is trained readily using gradient-based numerical optimization algorithms. The update rule is

$$\Lambda_t = \Lambda_{t-1} - \epsilon_t \frac{\partial L(\Lambda)}{\partial \Lambda}|_{\Lambda=\Lambda_{t-1}} \qquad (6)$$

where ϵ_t is the learning rate at the t^{th} iteration, Λ_t and Λ_{t-1} are the models at the t^{th} and the $(t-1)^{th}$ iteration, respectively.

2.4 Learning word embedding using DSSM

In [6], DSSM is learned on the clickthrough data. In this study, we train the DSSM on a big body of text. In order to train the DSSM, we first need to form the $< Q.D^+ >$ pairs. In this study, for each word in the train data, we take its neighboring words in sentences as the Q, and the word itself as the D^+. E.g., for the t-th word in a sentence, we form a training $< Q.D^+ >$ pair as:

$$Q = w_{t-d}, \dots w_{t-1}, w_{t+1}, w_{t+d}$$
$$D^+ = w_t$$

Then we can train the DSSM as described in the previous sections.

Comparing with related work, as discussed before, most previous work learns word embedding jointly with a language model, where the latter determines the loss function and thus is the optimization goal [14][15][16][17]. As a result, the optimization process is indirect with respect to the embedding layer. In contrast, DSSM produces the embedding vectors in the output layers and uses the cosine similarity between embedding vectors to construct the loss function. Therefore, it direct optimizes on the embedding learning layer. We found in a preliminary study that the DSSM approach is much easier to derive meaningful word embeding than Collobert and Weston [15]. Mnih and Kavukcuoglu [13] resembles our work in the aspect that it also optimizes the similarity between the embedding vectors of a word and its context. But their work represents the context embedding vector as a linear combination of the context word embedding vectors. In contrast, DSSM is a more general framework to project two sides of words into the same semantic space with deep structured neural network and optimize the similarity between vectors in that semantic space.

3 EXPERIMENTS

3.1 Implementation Details

We used the May 2013 dump of English Wikipedia as the dataset. We preprocessed Wikipedia by removing the formatting tags, mapping all words to lowercase, and breaking into sentences by a basic tokenizer. The most frequent 30,000 words consisting of only alphabet letters were taken as the vocabulary. (We decided not to include digits or punctuations in the vocabulary.) For every occurrence of a target inside-vocabulary word in every sentence in the dataset, we took an 11-word window from the sentence with the target word in the middle. We filtered out 11-word windows that contain more than one words outside the vocabulary; a special word "RARE" was used to replace those out-of-vocabulary words. Special words "PADDING" are added in the beginning and in the end of each sentence to form the 11-word windows when the target word is near the beginning or the end of the sentence. Finally we took 10 million such 11-word windows as the training dataset. The middle word will be D^+, and the remaining 10 context words will be Q, as discussed in the previous sections. For each pair of $< Q.D^+ >$, we selected 100 randomly words as $\{D_j^-; j = 1, \ldots, 100\}$.

Regarding the neural network structure, the query side (for the context words) consists of the word hashing layer, the non-linear hidden layer, and the output embedding layer; the document side (for the target word) consists of the word hashing layer and the output embedding layer. The word hashing layer has been discussed in Section 2.2. The hidden layer on the query side consists of 300 nodes, and the hidden layer is not used for the document side. The dimensionality of the learned word embedding is set to be 50, so both the output embedding layers consist of 50 nodes.

Mini-batch based stochastic gradient descent is used in the training stage, and each mini-batch consists of 1024 training samples. Training is conducted on a single Tesla K20 GPU. Each epoch takes roughly 210 seconds to run, and the total training time is about 2.4 hours (for 40 epochs).

3.2 Experimental Results

Firstly, we verify the quality of word embedding learning by visualization. The learned 50-dimensional embedding vectors are projected into 2-dimensional vectors using t-SNE [11] and plotted in Figure 1. It is well shown that words are clustered into multiple groups based on the synaptic and semantic meaning. Figure 1 illustrated three sample groups as being months, countries, and the US states. There also naturally exist clusters of verbs, adjectives, adverbs, and so on.

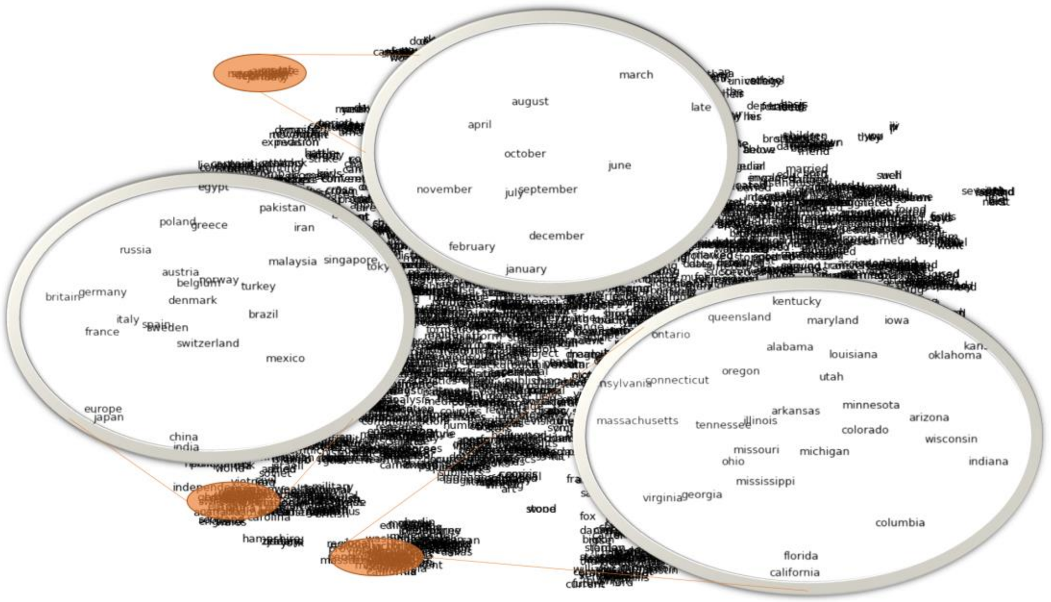

Figure 1 Plotting top 3,000 words in 2-D graph

Furthermore, we investigate the neighbor words for a few given words, in terms of cosine similarity of their embedding vectors. The result is shown in Table 1. We can see that semantically similar words have larger values of cosine similarity.

Table 1 Top three neighbor words (with cosine similarity) with respect to a few words

Word	Top 3 neighbor words		
king	earl (0.77)	pope (0.77)	lord (0.74)
women	person (0.79)	girl (0.77)	man (0.76)
france	spain (0.94)	italy (0.93)	belguim (0.88)
rome	constantinople (0.81)	paris (0.79)	moscow (0.77)
winter	summer (0.83)	autumn (0.79)	spring (0.74)
rain	rainfall (0.76)	storm (0.73)	wet (0.72)
car	truck (0.8)	driver (0.73)	motorcycle (0.72)

Lastly, we show that the learned word embedding vectors can be used to answer semantic analogy questions as in [12]. The analogy questions are of form "a is to b is as c is to ?". As discussed in [12], we first normalize all embedding vectors to unit norm. Then we take the embedding vectors x_a, x_b, x_c, and compute $y = x_b - x_a + x_c$. We find the word w^* whose embedding vector is closest to y according to cosine similarity:

$$w^* = \arg\max_{w} \frac{x_w^{\mathrm{T}} y}{||x_w||\,||y||}$$

Following Mnih and Kavukcuoglu [14], the word b or c are excluded from searching w^*.

Table 2 shows top three results for a few analogy questions with cosine similarity (between a candidate embedding vector x_w and the target vector y). We can see that the word embedding vectors indeed capture the semantic relationship between a pair of words by vector offsets. For example, the best result for the question "italy – rome = france – ?" is "paris", while neither "france" nor "rome" has "paris" as the closet word (Table 1). Take another example, the closest word for "winter" is "summer" (Table 1); when subtracted an offset ("summer – rain"), the closest word becomes "snow". Therefore, the learned embedding vectors capture the semantic relationship between words to some extent by vector offsets.

Table 2 Top three results for semantic analogy questions (with cosine similarity)

Word	Top 3 results		
man – woman = king – ?	mary (0.70)	prince (0.70)	queen (0.68)
italy – rome = france – ?	paris (0.78)	constantiople (0.74)	egypt (0.73)
summer – rain = winter – ?	snow (0.79)	rainfall (0.73)	wet (0.71)
man – eye = car – ?	motor (0.64)	overhead (0.58)	brake (0.58)
read – book = listen – ?	sequel (0.65)	tale (0.63)	song (0.60)

4 Conclusions

In this technical report, we proposed learning word embedding using DSSM. We show that the DSSM trained on large body of text can produce meaningful word embedding vectors as demonstrated on semantic word clustering and semantic word analogy tasks.

References

[1] Bengio, Y., 2009. “Learning deep architectures for AI,” Foundumental Trends Machine Learning, vol. 2, no. 1, pp. 1–127.

[2] Salakhutdinov R., and Hinton, G., 2007 “Semantic hashing,” in Proc. SIGIR Workshop Information Retrieval and Applications of Graphical Models.

[3] Hinton, G., and Salakhutdinov, R., 2010. “Discovering Binary Codes for Documents by Learning Deep Generative Models,” in Topics in Cognitive Science, pp 1-18.

[4] Gao, J., Toutanova, K., Yih., W-T. 2011. Clickthrough-based latent semantic models for web search. In SIGIR, pp. 675-684.

[5] Yih, W., Toutanova, K., Platt, J., and Meek, C. 2011. Learning discriminative projections for text similarity measures. In CoNLL.

[6] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T., and Harshman, R. 1990. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6): 391-407

[7] Dumais, S. T., Letsche, T. A., Littman, M. L., and Landauer, T. K. 1997. Automatic cross-linguistic information retrieval using latent semantic indexing. In AAAI-97 Spring Symposium Series: Cross-Language Text and Speech Retrieval.

[8] Li, P., Hastie, T., and Church, K.. 2006. "Very sparse random projections," in Proc. SIGKDD.

[9] Jarvelin, K. and Kekalainen, J. 2000. IR evaluation methods for retrieving highly relevant documents. In SIGIR, pp. 41-48.

[10] Huang, P., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data. In CIKM

[11] Maaten, L.J.P. van der and Hinton, G.E. 2008. Visualizing high-dimensional data using t-SNE. In Journal of Machine Learning Research 9(Nov):2579-2605

[12] Mikolov, T., Yih, W.-T. and Zweig, G. 2013. Linguistic regularities in continuous space word representations. In Proc. NAACL-HT.

[13] Mnih, A. and Kavukcuoglu, K. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In NIPS.

[14] Bengio, Y., Ducharme, R. and Vincent, P. 2003. A neural probabilistic language model. In Journal of Machine Learning Researc, 3:1137-1155.

[15] Collobert, R. and Weston, J. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In ICML.

[16] Huang, E., Socher, R., Manning, C. D. and Ng, A. Y. 2012. Improving word representations via global context and multiple word prototypes. In ACL.

[17] Mikolov, T., Chen, K., Corrado, G. and Dean, J. 2013. Efficient estimation of word representations in vector space. In ICLR.

[18] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J. and Khudanpur, S. 2010. Recurrent neural network based language model. In Interspeech.