Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Pre-Trained Multi-View Word Embedding Using Two-side Neural Network

Yong Luo[†*], **Jian Tang**[†], **Jun Yan**[‡], **Chao Xu**[†], **and Zheng Chen**[‡]
[†]School of Electronics Engineering and Computer Science, Peking University, Beijing, China
(email: yluo180@gmail.com; tangjianpku@gmail.com; xuchao@cis.pku.edu.cn)
[‡]Microsoft Research Asia, Beijing, China (email: {junyan, zhengc}@microsoft.com)

Abstract

Word embedding aims to learn a continuous representation for each word. It attracts increasing attention due to its effectiveness in various tasks such as named entity recognition and language modeling. Most existing word embedding results are generally trained on one individual data source such as news pages or Wikipedia articles. However, when we apply them to other tasks such as web search, the performance suffers. To obtain a robust word embedding for different applications, multiple data sources could be leveraged. In this paper, we proposed a two-side multimodal neural network to learn a robust word embedding from multiple data sources including free text, user search queries and search click-through data. This framework takes the word embeddings learned from different data sources as pre-train, and then uses a two-side neural network to unify these embeddings. The pre-trained embeddings are obtained by adapting the recently proposed CBOW algorithm. Since the proposed neural network does not need to re-train word embeddings for a new task, it is highly scalable in real world problem solving. Besides, the network allows weighting different sources differently when applied to different application tasks. Experiments on two real-world applications including web search ranking and word similarity measuring show that our neural network with multiple sources outperforms state-of-the-art word embedding algorithm with each individual source. It also outperforms other competitive baselines using multiple sources.

Introduction

Word embedding is a continuous-valued representation of the word. Good word embedding is expressive and effective since it can represent a huge number of possible inputs using only a small number of variables, and help tackling the problem of the curse of dimensionality. By representing each word with the learned continuous-valued variables, semantically related words can be close to each other. The effectiveness of word embedding has been investigated in the literature, such as (Bengio, Courville, and Vincent 2013). Word embedding has been widely used in many natural language processing (NLP) tasks, such as language modeling (Bengio et al. 2003; Mnih and Hinton 2007), named entity recognition and chunking (Collobert and Weston 2008; Turian, Ratinov, and Bengio 2010; Dhillon, Foster, and Ungar 2011). Recently, the word embedding techniques were also extended to embed queries, documents, phrases, entities, etc. (Huang et al. 2013; Mikolov et al. 2013b), which could play a critical role in industry applications, such as large-scale web search and knowledge mining.

Dozens of algorithms have been proposed for word embedding in the literature. Some representative works are the Collobert and Weston (2008) (C&W08) (Collobert and Weston 2008) method, hierarchical log-bilinear (HLBL) (Mnih and Hinton 2008) model, and the continuous bag-of-words (CBOW) (Mikolov et al. 2013a) algorithm. In the training of these models, the data used are articles from books, news or Wikipedia etc., which are all consist of large amounts of sentences. Usually the context information contained in and between the sentences is utilized for training. Such information is critical and effective for the NLP tasks, but may be not appropriate for other tasks such as query classification (Lin and Wu 2009).

To tackle this problem, we propose to train word embedding for different application tasks by the use of different data sources, such as the free text documents, user queries, and click-through data. Besides, it has been demonstrated empirically in (Turian, Ratinov, and Bengio 2010) that combining different word embeddings is beneficial. A natural combination strategy is to concatenate all the embeddings into a long vector. However, this strategy not only lack physical interpretation, but also may lead to serve "curse of dimensionality". We thus developed a two-side neural network to combine the learned word embeddings. In particular, we first introduce how to adapt a word embedding algorithm to learn from other data sources that have never been reported in academia before. Due to the efficiency and effectiveness of the CBOW model, we use it as our baseline word embedding algorithm, and show how to train it on different data sources. Then we take all the learned word embeddings as the pre-train of a two-side neural network, and only fine tune the hidden layer parameters for different application tasks. The network is chosen to have two sides since we found that the data is pairwise in many applications, e.g., the query doc-

[*]This work was done when the first author and second author were interns of Microsoft Research Asia.

ument pair in web search, the word pair in word similarity measuring, and the context target pair in language modeling.

The main advantages of the proposed multi-view word embedding (MVWE) framework are: 1) different from the work of (Collobert et al. 2011), where word embeddings are pre-trained and fine tuned for each NLP task, the multiple pre-trained word embeddings in our framework are fixed, and we only need to fine tune the combination weights when apply MVWE to different tasks. This reduces the time complexity significantly and thus is scalable to a variety of real world applications; 2) compared to the combination strategy presented in (Turian, Ratinov, and Bengio 2010), where word embeddings are only utilized as additional features in the NLP system, the hidden units in our network are abstractions of the inputs (Bengio 2009) and thus the commonalities as well as the variations of different sources can be exploited using the multiple hidden layers. Besides, the neural network can approximate the arbitrary nonlinear correlation among the different sources (Krasnopolsky and Lin 2012), and thus hopefully better performance could be obtained.

We thoroughly evaluate the proposed MVWE framework on two different application tasks: search ranking and semantic word similarity measuring. We compare it with single word embedding trained on each data source, as well as a combination of these embeddings using the concatenation strategy and linear regression. Experimental results demonstrated the effectiveness of the proposed framework.

Related Work

Our work is a combination of the word embeddings pre-trained from multiple views.

Word embedding

Traditional one-hot word representation, in which the dimensionality of the word vector is the same as the size of the dictionary, suffers the data sparsity problem. Therefore, many researchers focus on representing a word using a continuous-valued low-dimensional feature vector (Dumais et al. 1988; Brown et al. 1992; Bengio et al. 2003). Word embedding (Bengio et al. 2003; Collobert and Weston 2008; Mnih and Hinton 2008) is one of the most popular word representations of this type. We refer to (Turian, Ratinov, and Bengio 2010) for a summarization of some popular word representation works. Recently, two quite efficient models, continuous bag-of-words (CBOW) and continuous skip-gram (Skip-gram) are proposed in (Mikolov et al. 2013a). High-quality word embeddings can be learned using these two models and the training process was further accelerated in (Mikolov et al. 2013b) by sub-sampling the frequent words. Due to the efficiency and effectiveness of these two models, we use one of them, the CBOW model, as our baseline word embedding algorithm. We will show how to adapt it for training on various data sources, and then combine the learned word embeddings for different applications.

To the best of our knowledge, the closest work to our model is done by Turian et al. (Turian, Ratinov, and Bengio 2010). Compared to their work, our framework offers at least two benefits: 1) the input word embeddings are pre-trained using different data sources, not different algorithms. Thus our model can be applied to a variety of tasks including web search, not limited in NLP tasks; 2) we combine the different word embeddings using a nonlinear network, which can be fine tuned for specific task. While in (Turian, Ratinov, and Bengio 2010), the word embeddings were only used as additional features in the supervised NLP system and thus the nonlinear relationships among the different embeddings cannot be exploited.

Multi-view learning using the neural network

The present framework is also closely related to multimodal fusion (Luo et al. 2013a; 2013b). The different embeddings of a word can be regarded as its different modalities since the embeddings are learned from different data sources. We refer to (Kung and Hwang 1998) for a survey of some representative works on combining multiple modalities using neural network. There also exist a few recent models based on deep networks. Ngiam et al. (Ngiam et al. 2011) proposed to fuse the audio and video inputs for speech classification using a Deep Autoencoder model. As opposed to it, a bimodal Deep Boltzmann Machine (DBM) is presented in (Srivastava and Salakhutdinov 2012) to integrate together the image and text modalities. The multimodal DBM is generative and thus can naturally handle missing data modalities. In this paper, we assume the word embeddings are pre-trained, and our aim is to combine them. The two multimodal deep models are not appropriate here since they are not feasible when only the trained embeddings are available.

Multi-view Word Embedding

In this section, we first illustrate how to train a word embedding algorithm using different data sources, and then present the main part of this paper, i.e., the multi-view word embedding (MVWE) framework on combining the word embeddings trained on different data sources for various application tasks.

Word embedding algorithm adaptation

The recently proposed high-performance CBOW algorithm (Mikolov et al. 2013a) is utilized as an example for adaptation, and the data sources we used here are: free text documents from Wikipedia, search click-through data, and user query data collected using a commercial search engine. In CBOW, a negative sampling (NEG) (Mikolov et al. 2013b) training strategy is proposed. NEG can learn high-quality word embeddings without maximizing the probability of the target word in an n-gram sequence over the vocabulary.

For the free text documents, CBOW can be directly used as usual, and in our implementation we consider a window size of $c = 11$ words. However, the window size must be limited to $c = 3$ for user query data, since there is sometimes only one word (together with two "padding" words (Collobert et al. 2011)) for a query. Besides, a window should not across two queries since they are often not semantically related.

For the search click-through data, all the query words are used as the context words, and the clicked document is regarded as the target word. The document embedding can be

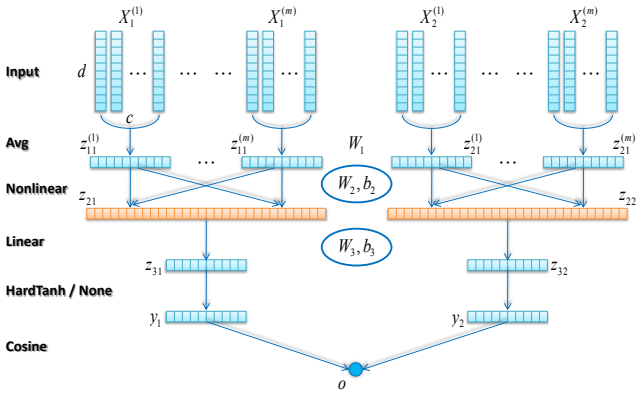


Figure 1: Architecture of the MVWE network.

pre-trained by the deep structured semantic model (DSSM) presented in (Huang et al. 2013). The same strategy can be applied in many other popular word embedding learning algorithms, such as the C&W08 (Collobert and Weston 2008) and log-bilinear (LBL) (Mnih and Hinton 2007) model, by simply replacing the target word embedding with the pre-trained document embedding. We only adapt the CBOW algorithm in this paper due to its scalability, efficiency and high performance.

Network for word embedding combination

In this paper, we choose the neural network to combine the different word embeddings. This is mainly because the nonlinear combination is often superior the linear combination, and the neural network can approximate arbitrary nonlinear dependence among multiple modalities (Krasnopolsky and Lin 2012). Before presenting the network, we first introduce some notations.

Notations: We consider a two-side neural network of L layers with parameters θ. Given a sequence of c input words, we stack their embeddings in a matrix $X_s^{(i)} \in \mathbb{R}^{d_i \times c}$, where $s \in \{1, 2\}$ and $i \in \{1, \ldots, m\}$ are the index of the side and data source respectively, m signifies the number of data sources and d_i is the dimensionality of the word embedding pre-trained from the i'th data source. We denote z_{js} as the representation of the j'th hidden layer. At the first hidden layer, there are m representations $\{z_{1s}^{(1)}, \ldots, z_{1s}^{(m)}\}$ and z_{1s} is a concatenation of them. We also denote y_s as the final unified word embedding of the inputs $X_s^{(i)}, i = 1, \ldots, m$, and o as the output of the network.

The two-side network for combination is depicted in Figure 1. For each side of the network, the input is a word sequence. We map the word sequence into a feature matrix $X_s^{(i)}$ according to the lookup table trained by CBOW on the i'th data source. Each column of $X_s^{(i)}$ is the embedding of one word in the sequence. Then we obtain m feature matrix $X_s^{(1)}, \ldots, X_s^{(m)}$ for each side, and each $X_s^{(i)}$ is averaged by column (i.e., the embeddings of the different words in a sequence are averaged) to obtain the first hidden layer representation $z_{1s}^{(i)}$. All the $z_{1s}^{(i)}, i = 1, \ldots, m$ are concatenated as z_{1s}, which is subsequently transformed into the last hidden layer representation z_{3s} by going through a nonlinear followed by a linear layer. The average layer has no parameters to be learned, and the nonlinear and linear layers are supposed to be parameterized by $\{W_2, b_2\}$ and $\{W_3, b_3\}$ respectively, where W_i is the weight matrix and b_i is the bias term. Then we have

$$z_{2s} = f(W_2 z_{1s} + b_2), \text{and } z_{3s} = W_3 z_{2s} + b_3, \quad (1)$$

where f is a nonlinear activation function. Following (Collobert et al. 2011), the hard version of $tanh$ is adopted, i.e.,

$$f(x) = \begin{cases} -1, & \text{if } x < -1 \\ x, & \text{if } -1 \leq x \leq 1 \\ 1, & \text{if } x > 1 \end{cases} \quad (2)$$

The nonlinear hidden layer is optional, and sometimes we can remove it from the network for the sake of efficiency. The hard $tanh$ operation on z_{3s} is also optional, so the unified embedding of the input sequence can be either $y_s = z_{3s}$ or $y_s = f(z_{3s})$. Finally, we utilize the cosine function to measure the similarity between the two sides of the network, that is,

$$o = \text{cosine}(y_1, y_2) = \frac{y_1^T y_2}{\|y_1\| \|y_2\|}. \quad (3)$$

In the following, we show how to train the network for different tasks. It should be noted that we only fine tune the parameters $\theta = \{W_i, b_i\}$, and the input word embeddings are fixed.

Training

Our neural network is trained by leveraging the negative sampling (NEG) (Mikolov et al. 2013b) strategy for the search ranking task. In the semantic word similarity task, we proposed a pairwise distance regression (PADR) training strategy.

NEG for search ranking NEG is to separate the target element e_O in an n-gram sequence from some noise elements $e_i, i = 1, \ldots, k$ drawn according to a noise distribution $P_n(e)$. For search ranking, the n-gram context is the query words and the target element is the clicked document. The noise elements are drawn from the document corpus. Suppose that y_{e_C} and y_{e_O} are the unified embedding of the context and target element respectively, then the objective of NEG (Mikolov et al. 2013b) is given by

$$P(\theta) = \log\sigma(y_{e_C}^T y_{e_O}) + \sum_{i=1}^{k} \mathbb{E}_{e_i \sim P_n(e)}[\log\sigma(-y_{e_C}^T y_{e_i})], \quad (4)$$

where y_{e_i} is the unified embedding of the noise element e_i, and $\sigma(x) = 1/(1 + \exp(-x))$. In our implementation, all y_{e_C}, y_{e_O} and y_{e_i} are normalized to compute the output score o by replacing the inner products $y_{e_C}^T y_{e_O}$ and $y_{e_C}^T y_{e_i}$ in (4) with the cosine similarities.

PADR for computing semantic word similarity We argue that the semantic similarity of a word pair is relative. For example, compared to {cat, car}, the words {cat, bird}

Table 1: Word embeddings in the lookup table trained by different data sources. Each column is the queried word followed by its 10 most similar words in the dictionary.

word			wikipedia			megabits		
click	query	wiki	click	query	wiki	click	query	wiki
Word	words	phrase	wiki	wiki	Wiki	mbits	kilobits	kib
WORD	powerpoint	words	WIKI	site:wikipedia.org	nupedia	megabit	nanograms	megabytes
www.word	palana	suffix	encyclopedia	is	wikinfo	kilobits	mbytes	terabytes
words	Tecate	term	site:wikipedia.org	does	slashdot	mbytes	ug/l	gigabytes
word.com	murase	verb	site:en.wikipedia.org	facts	wiktionary	gbytes	mg/l	kilobytes
msword	VMotion	digraph	Wiki	vs	wikiquote	megabytes	MPa	kilowatts
wordgames	worksheets	copula	Wikipedia	wikia	npov	gigabits	.jpg	milligrams
vocabulary	Jeanna	possessive	WIKIPEDIA	bio	deletion	Mb	mi2	liter
thesaurus	wordsearch	acronym	WIki	define:	fark	megabyte	dvr-ms	litres
word?	PERSIANS	tetragrammation	wki	website	templates	kbytes	mbar	nanometres

are semantic similar to each other since they are both animals. But compared to {cat, tiger}, {cat, bird} are not such close as they indicate different animal families. Therefore, we propose to target the similarity difference of two word pairs, instead of the similarity of each word pair. The loss of the proposed pairwise distance regression (PADR) strategy is given by

$$L(\theta) = \frac{1}{2}(o_{ij} - t_{ij})^2, \tag{5}$$

where $o_{ij} = o_i - o_j$ is the difference of the output similarity between the i'th word pair and j'th pair, and t_{ij} is the corresponding ground-truth similarity difference.

Stochastic gradient In the learning process, we maximize (4) or minimize (5) with stochastic gradient. In each iteration, a random sample is selected to make a gradient step: $\theta \leftarrow \theta + \lambda \frac{\partial P(\theta)}{\partial \theta}$ or $\theta \leftarrow \theta - \lambda \frac{\partial L(\theta)}{\partial \theta}$, where λ is the learning rate, which is determined adaptively following the implementation of CBOW[1]. That is, λ is start with a large value, e.g., 0.25, and then decreases with the number of iterations until it is smaller than a threshold. This strategy was observed to be better than using a fixed learning rate.

In our implementation, we set the dimensionalities of all the pre-trained word embeddings and the final unified word embedding to be 192. Thus the first hidden layer size is also 192. The middle hidden layer size is parameter that can be decided using grid-search. The initialization of the network is according to (Montavon, Orr, and Muller 2012). Besides, in search ranking, the document embedding pre-trained by DSSM (Huang et al. 2013) is directly used here as y_2 in our network. Thus we only have to train one side of network in Figure 1.

Experiments

In the experiments, we first show a case study of the word embeddings learned from different data sources, and then evaluate the proposed multi-view word embedding (MVWE) algorithm on two diverse application tasks: search ranking and semantic word similarity measuring.

[1]http://code.google.com/p/word2vec/

Case study of word embedding trained on individual data source

To pre-train the click-through and user query word embeddings, we use a click log training set that includes about 200 million English queries (3 words for each query on average), each associated with a clicked document (URL). The obtained results are called CBOW_click and CBOW_query respectively. The data used to train word embedding on free text is the first billion characters from Wikipedia[2], and we call the learned word embedding CBOW_wiki.

Table 1 shows the ten most similar words of a few randomly chosen query words, under the metric induced by different word embeddings. We find from the results that: 1) almost all the similar words returned by CBOW_click is semantically close to the query words, although some are websites or only have spell difference. The phenomenon was observed even if more similar words are listed (not shown due to the limited space). This is because users' intention and preference has been contained in the learned word embeddings; 2) CBOW_wiki only tends to return the words that belong to the same class of the query word; 3) CBOW_query can capture users intention to some extent, but most of the returned nearest words are not related to the query. Nevertheless, it can find some meaningful words missed in the other two embeddings, e.g., "powerpoint" for the query "word".

Overall, the word embeddings trained on different sources could be suitable to different tasks. For example, CBOW_click is good for web search, while CBOW_wiki may be appropriate for word clustering. In the next, we show the results of combining these three word embeddings for specific tasks using the proposed MVWE framework compared with several baselines.

Experimental setup

In this paper, we assume the word embeddings are pre-trained, and our aim is to combine them. We thus only compared to the embedding combination strategies such as concatenation and regression. Specifically, we compared MVWE with the following methods in both of the investigated application tasks.

[2]http://mattmahoney.net/dc/textdata.html

• **Single word embedding**: using the word embeddings trained on a single type of data. Particularly, in search ranking, the query embedding is an average of the query words' embeddings. It should be noted that the training of CBOW_click is supervised in search ranking since the click-through information is utilized, while for CBOW_query and CBOW_wiki, the training are both unsupervised.
• **CONCAT**: singly concatenate the different types of word embeddings into a long vector, which is used in the same way as the single word embedding.
• **REG**: combine the word embeddings by the use of linear regression, provided that the dimensionalities of different embeddings are the same. For search ranking, suppose there are N training samples, and the averaged context embeddings of different data sources are stacked in a matrix $Z_i = [z_i^{(1)}, \ldots, z_i^{(m)}] \in \mathbb{R}^{d \times m}$ for the i'th sample. Then the objective in search ranking is to minimize $L(\alpha) = \frac{1}{2N} \sum_{i=1}^{N} \|Z_i\alpha - y_i\|_2^2 + \frac{\gamma}{2}\|\alpha\|_2^2$ w.r.t. α, where y_i is the pre-trained document embedding, $\alpha \in \mathbb{R}^m$ is the regression variable to be learned, and γ is a positive trade-off parameter. The problem can be solved analytically. Then the query embedding is a linear combination of the averaged word embeddings trained on different data sources with the learned weight α.

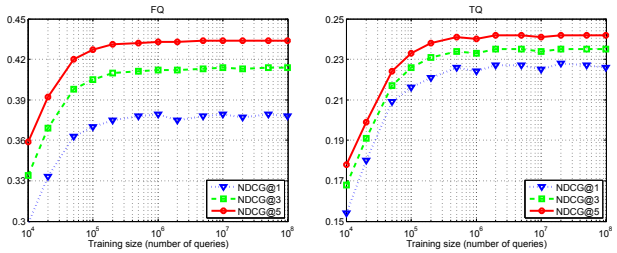
For word similarity measuring, the objective is to minimize $\alpha^T H\alpha - t$ w.r.t α, where $H = \frac{2}{N(N-1)} \sum_{i<j} H_{ij}$ with each $H_{ij} = Z_{i1}^T Z_{i2} - Z_{j1}^T Z_{j2}$, and $t = \frac{2}{N(N-1)} \sum_{i<j} t_{ij}$ with each t_{ij} the ground-truth similarity difference of the i'th and j'th word pair. One solution of this problem is $\alpha = \sqrt{t/\lambda}u$. Here λ and u are the largest eigenvalue and the corresponding eigenvector of H.
• **AVG**: the same as REG, only the weights are the same for different data sources.

For MVWE, the pre-trained document embedding is directly utilized as y_2 in search ranking. In word similarity measuring, the average layer can be omitted actually since there is only one word for each sequence.

Search ranking

The goal of search ranking is to compute relevance scores between a given query and a set of documents, and then rank the documents according to these scores. The training set is the same as in training CBOW_click. We test the trained model on another click log dataset, from which we randomly sampled 550K frequent queries (FQ), and 600K long tail queries (TQ) as two subsets. All the data are collected using a commercial search engine. The relevance between the query and document are assigned with 5 levels: perfect, excellent, good, fair, and bad. The performance is evaluated in terms of normalized discounted cumulative gain (NDCG) (Järvelin and Kekäläinen 2000), and the NDCG scores at the 1, 3, 5 truncation levels are reported.

We first evaluate MVWE by varying the size of the training set. The nonlinear hidden layer size is set to be 576=192×3, which is the sum of the input word embedding sizes. The experimental results are shown in Figure 2. It can be seen that: 1) the NDCG score at higher truncation level tends to be larger; 2) the performance improves at first



(a) (b)

Figure 2: Effect of training size (number of queries) on the two test sets for search ranking.

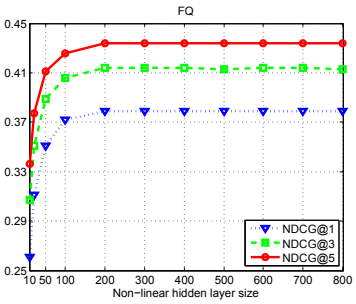


Figure 3: Effect of nonlinear hidden size for search ranking.

with an increasing training size, and then approach to a fixed value when the number of queries used is more than 500K. This indicates that the training time cost of our model is low, since not very large amount of training data are required to achieve the best performance.

Then we analyze the parameters of our model. Here, only the impact of nonlinear hidden layer size is investigated, as we have fixed the size of other layers and used an adaptive learning rate. The training size is set to be 10M since it is observed to perform the best overall in the last experiments. The results are shown in Figure 3. It can be observed that the performance improves sharply at first when the number of hidden units increases, and then almost keeps unchanged after 200. This again indicates the low cost of the training process, since only a small number of nonlinear hidden units are required for training.

Finally we compare MVWE with the baselines. The results are presented in Table 2. It can be seen that: 1) the supervised CBOW_click is much better than the unsupervised CBOW_query and CBOW_wiki. This demonstrates the effectiveness of the word embedding algorithm adaptation strategy; 2) CBOW_query is superior to CBOW_wiki on the FQ set, and they are comparable on the TQ set. This is because the user query data capture users intention, and thus is more informative for search than the arbitrary free text; 3) the simple concatenation strategy CONCAT is better than CBOW_query and CBOW_wiki, but much worse than the best performed single word embedding CBOW_click; 4) the average of the three types of input word embeddings AVG is also worse than CBOW_click, while the linear regression

Table 2: NDCG performance of different methods for search ranking.

	FQ (NDCG)			TQ (NDCG)		
Methods	@1	@3	@5	@1	@3	@5
CBOW_click	0.362	0.393	0.412	0.217	0.227	0.234
CBOW_query	0.145	0.180	0.208	0.077	0.090	0.102
CBOw_wiki	0.136	0.169	0.194	0.079	0.092	0.103
CONCAT	0.187	0.208	0.225	0.099	0.108	0.118
REG	**0.380**	**0.414**	**0.434**	0.217	0.226	0.234
AVG	0.350	0.385	0.407	0.195	0.205	0.213
MVWE	**0.379**	**0.414**	**0.434**	**0.226**	**0.234**	**0.241**

strategy REG that learns the combination weights can outperform CBOW_click significantly on the FQ set. However, no improvement is observed on the TQ set. This indicates that the simple linear combination is effective sometimes, but not stable. In contrast, the proposed MVWE outperforms CBOW_click consistently on both test sets. In particular, we obtain a 4.7%, 5.3% and 5.3% NDCG improvement at the truncation level 1, 3, and 5, respectively on the FQ set.

To sum up, CBOW_click is the most effective individual word embedding for search ranking, and better performance can be obtained by combining all the embeddings using linear regression and the proposed MVWE algorithm. The concatenation strategy fails in this application.

Word similarity measuring

We use the WordSim353 (Finkelstein et al. 2002) dataset, one of the most popular collection of this kind, to evaluate our model for computing semantic word relatedness. Both the Pearson's linear correlation coefficient (score based correlation) and the Spearman's rank correlation coefficient (rank based correlation) are utilized as the evaluation criteria (Strube and Ponzetto 2006; Gabrilovich and Markovitch 2007).

The WordSim353 dataset contains 353 word pairs, and the similarity of each word pair is an average of the scores assigned by 13-16 subjects. We randomly sampled 300 word pairs as the training set, and the remaining 53 word pairs are used for test. We removed the nonlinear hidden layer from the network due to the limited training data, and actually the result is bad (data not shown) when it is added.

Similar as in the search ranking task, we first perform a self-test of our model. The results are shown in Figure 4. We can see that both the Pearson and Spearman correlation improve with an increasing number of word pairs used in general. The tendency indicates a better performance if more training data are available.

Then we compare MVWE with the baselines. We report the results in Table 3, where MVWE_n means the training size is n. It can be observed that: 1) CBOW_click is superior to both CBOW_query and CBOW_wiki. This indicates that the click-through information is useful for inducing semantic word similarity; 2) the performance of CBOW_query is worse than CBOW_wiki since the semantic meaning of the word cannot be captured due to the lack of structure and context information; 3) the CONCAT strategy is very

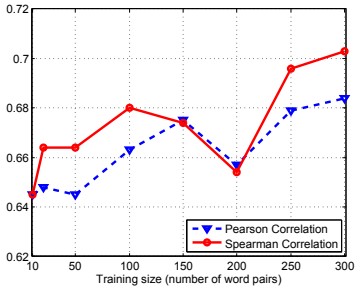


Figure 4: Effect of training size (number of word pairs) for word similarity measuring.

Table 3: Comparsion of different methods for word similarity measuring.

Methods	Pearson Correlation	Spearman Correlation
CBOW_click	0.594	0.594
CBOW_query	0.483	0.484
CBOw_wiki	0.559	0.572
CONCAT	0.650	0.679
REG	0.534	0.529
AVG	0.531	0.580
MVWE_100	0.663	0.680
MVWE_300	**0.684**	**0.703**

competitive in this task and outperforms the other baselines significantly; 4) the REG strategy performs worse than CBOW_click and CBOW_wiki, and thus fails to combine the different word embeddings. This again indicates that the simple linear combination strategy is unreliable for word embedding combination. In contrast, the MVWE framework tends to outperform all of the other methods if the training size is more than 100, and the improvement is significantly when all the training word pairs are utilized.

To sum up, CBOW_click and CBOW_wiki are both effective for word similarity measuring, and we can obtain much better performance using the concatenation and MVWE methods. The linear regression strategy fails in this application.

Conclusion

This paper presents a two-side neural network architecture that can integrate multiple word embeddings for very different application tasks, such as search ranking and semantic word relatedness measuring. The input word embeddings are pre-trained by adapting the existed word embedding algorithm for different data sources. The network is able to be fine tuned for the tasks of interest, and also output a unified word embedding for each of them. The training cost of the network is low since neither much training data nor large nonlinear hidden layer size is needed. The performance of the concatenation and linear regression strategy, which may fail on some tasks, are not robust for combining word embeddings. In contrast, the proposed framework outperforms the baselines consistently for all the investigated tasks. The future work may be to introduce more word embedding al-

gorithms for adaptation and combination, and include more application tasks for verification.

Acknowledgments

This work is partially supported by NBRPC 2011CB302400, NSFC 60975014, 61121002, JCYJ20120614152136201, NSFB 4102024.

References

Bengio, Y.; Ducharme, R.; Vincent, P.; and Jauvin, C. 2003. A neural probabilistic language model. *Journal of Machine Learning Research* 3:1137–1155.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8):1798–1828.

Bengio, Y. 2009. Learning deep architectures for ai. *Foundations and trends in Machine Learning* 2(1):1–127.

Brown, P. F.; Desouza, P. V.; Mercer, R. L.; Pietra, V. J. D.; and Lai, J. C. 1992. Class-based n-gram models of natural language. *Computational Linguistics* 18(4):467–479.

Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning*, 160–167.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research* 12:2493–2537.

Dhillon, P.; Foster, D. P.; and Ungar, L. H. 2011. Multi-view learning of word embeddings via cca. In *Advances in Neural Information Processing Systems*, 199–207.

Dumais, S. T.; Furnas, G. W.; Landauer, T. K.; Deerwester, S.; and Harshman, R. 1988. Using latent semantic analysis to improve access to textual information. In *SIGCHI conference on Human factors in computing systems*, 281–285.

Finkelstein, L.; Gabrilovich, E.; Matias, Y.; Rivlin, E.; Solan, Z.; Wolfman, G.; and Ruppin, E. 2002. Placing search in context: The concept revisited. *ACM Transactions on Information Systems* 20(1):116–131.

Gabrilovich, E., and Markovitch, S. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *International Joint Conference on Artificial Intelligence*, 1606–1611.

Huang, P.-S.; He, X.; Gao, J.; Deng, L.; Acero, A.; and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data. In *ACM international conference on Conference on Information & Knowledge Management*, 2333–2338.

Järvelin, K., and Kekäläinen, J. 2000. Ir evaluation methods for retrieving highly relevant documents. In *ACM SIGIR conference on Research and development in information retrieval*, 41–48.

Krasnopolsky, V. M., and Lin, Y. 2012. A neural network nonlinear multimodel ensemble to improve precipitation forecasts over continental us. *Advances in Meteorology* 2012(doi:10.1155/2012/649450).

Kung, S.-Y., and Hwang, J.-N. 1998. Neural networks for intelligent multimedia processing. *Proceedings of the IEEE* 86(6):1244–1272.

Lin, D., and Wu, X. 2009. Phrase clustering for discriminative learning. In *Joint Conference of the Annual Meeting of the ACL and the International Joint Conference on Natural Language Processing*, 1030–1038.

Luo, Y.; Tao, D.; Xu, C.; Li, D.; and Xu, C. 2013a. Vector-valued multi-view semi-supervised learning for multi-label image classification. In *AAAI Conference on Artificial Intelligence*, 647–653.

Luo, Y.; Tao, D.; Xu, C.; Liu, H.; and Wen, Y. 2013b. Multiview vector-valued manifold regularization for multilabel image classification. *IEEE Transactions on Neural Networks and Learning Systems* 24(5):709–722.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. In *ICLR Workshop*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 3111–3119.

Mnih, A., and Hinton, G. 2007. Three new graphical models for statistical language modelling. In *International Conference on Machine Learning*, 641–648.

Mnih, A., and Hinton, G. E. 2008. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems*, 1081–1088.

Montavon, G.; Orr, G. B.; and Muller, K.-R. 2012. *Neural networks: tricks of the trade (2nd edition)*. Springer.

Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; and Ng, A. 2011. Multimodal deep learning. In *International Conference on Machine Learning*, 689–696.

Srivastava, N., and Salakhutdinov, R. 2012. Multimodal learning with deep boltzmann machines. In *Advances in Neural Information Processing Systems*, 2231–2239.

Strube, M., and Ponzetto, S. P. 2006. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI Conference on Artificial Intelligence*, 1419–1424.

Turian, J.; Ratinov, L.; and Bengio, Y. 2010. Word representations: a simple and general method for semi-supervised learning. In *Annual Meeting of the Association for Computational Linguistics*, 384–394.