Chapter 4
Word Embedding for Understanding Natural Language: A Survey

Yang Li and Tao Yang

4.1 Introduction

Natural language understanding from text data is an important field in Artificial Intelligence. As images and acoustic waves can be mathematically modeled by analog or digital signals, we also need a way to represent text data in order to process it automatically. For example, the sentence "The cat sat on the mat." can not be processed or understood directly by the computer system. The easiest way is to represent is through a sparse discrete vector $\{(i_{cat}, 1), (i_{mat}, 1), (i_{on}, 1), (i_{sit}, 1), (i_{the}, 2)\}$, where i_w denotes the index of word w in the vocabulary. This is called one-hot embedding. However, there are several disadvantages for this simple model. First, it generates high dimensional vectors whose length depends on the volume of the vocabulary, which is usually very large. Meanwhile, the semantic relationship between words (e.g., "sit on") cannot be reflected by these separate counts. Due to the subjectivity of languages, the meaning of word (phrase) varies in different contexts. This makes it a more challenging task to automatically process text data.

A goal of language modeling is to learn the joint probability that a given word sequence occurs in texts. A larger probability value indicates that the sequence is more commonly used. One candidate solution to involve the relationship between words is called "n-gram" model, where n is the hyperparameter manually chosen. The "n-gram" model takes into consideration the phrases of n consecutive words. In the example above, "the cat", "cat sat", "sat on", "on the", "the mat" will be counted, if we set $n = 2$. The disadvantage of "n-gram" model is that it is constrained by the parameter n. Moreover, the intrinsic difficulty of language modeling is that: a word sequence that is used to test the model is likely to be different from all the sequences in the training set (Bengio et al. 2003). To make it more concrete, suppose

Y. Li • T. Yang (✉)
School of Automation, NorthWestern Polytechnical University, Xi'an, Shanxi 710072, P.R. China
e-mail: liyangnpu@mail.nwpu.edu.cn; yangtao107@nwpu.edu.cn

S. Srinivasan (ed.), *Guide to Big Data Applications*, Studies in Big Data 26,
DOI 10.1007/978-3-319-53817-4_4

we want to model the joint distribution of 5-word sequences in a vocabulary of 100,000 words, the possible number of combinations is $100000^5 - 1 = 10^{25} - 1$, which is also the number of free parameters to learn. It is prohibitively large for further processing. This phenomenon is called "the curse of dimensionality". The root of this curse is the "generalization" problem. We need an effective mechanism to extrapolate the knowledge obtained during training to the new cases. For discrete spaces mentioned above, the structure of generalization is not obvious, i.e. any change on the discrete variables, as well as their combinations, will have a drastic influence on the value of joint distribution to be estimated.

To solve the dilemma, we can model the variables in a continuous space, where we can think of how training points trigger probability mass and distribute it smoothly to the neighborhood around them. Then it goes back to the problem of word embedding, whose concept was first introduced in Hinton (1986). Word embedding, sometimes named as word representation, is a collective name for a set of language models and feature selection methods. Its main goal is to map textual words or phrases into a low-dimensional continuous space. Using the example above, "cat" can be denoted as $[0.1, 0.3, \ldots, 0.2]^M$ and "mat" can be expressed as $[0.1, 0.2, \ldots, 0.4]^M$, where M is the hyperparameter. After that, advanced NLP tasks can be processed implemented based on these real-valued vectors. Word embedding encodes the semantic and syntactic information of words, where semantic information mainly correlates with the meaning of words, while syntactic information refers to their structural roles. Is a basic procedure in natural language processing. From the high level, most of the models try to optimize a loss function trying to minimize the discrepancy between prediction values and target values. A basic assumption is that words in similar context should have similar meaning (Harris 1954). This hypothesis emphasizes the bound of $(w, \tilde{w})$ for common word-context pairs (word w and its contextual word $\tilde{w}$ usually appear together) and weaken the correlation of rare ones.

The existing word embedding approaches are diverse. There are several ways to group them into different categories. According to Sun et al. (2015) and Lai et al. (2015), the models can be classified as either paradigmatic models or syntagmatic models, based on the word distribution information. The text region where words co-occur is the core of the syntagmatic model, but for the paradigmatic model it is the similar context that matters. Take "The tiger is a fierce animal." and "The wolf is a fierce animal." as examples, "tiger-fierce" and "wolf-fierce" are the syntagmatic words, while "wolf-tiger" are the paradigmatic words. Shazeer et al. (2016) divides the models into two classes, matrix factorization and slide-window sampling method, according to how word embedding is generated. The former is based on the word co-occurrence matrix, where word embedding is obtained from matrix decomposition. For the latter one, data sampled from sliding windows is used to predict the context word.

In this chapter, word embedding approaches are introduced according to the method of mapping words to latent spaces. Here we mainly focus on the Neural Network Language Model (NNLM) (Xu and Rudnicky 2000; Mikolov et al. 2013) and the Sparse Coding Approach (SPA) (Yogatama et al. 2014a). Vector Space

Model aims at feature expression. A word-document matrix is first constructed, where each entry counts the occurrence frequency of a word in documents. Then embedding vectors containing semantic information of words are obtained through probability generation or matrix decomposition. In the Neural Network Language Model, fed with training data, word embedding is encoded as the weights of a certain layer in the neural network. There are several types of network architectures, such as Restricted Boltzmann Machine, Recurrent Neural Network, Recursive Neural Network, Convolutional Neural Network and Hierarchical Neural Network. They are able to capture both the semantic and syntactic information. Sparse coding model is another state-of-the-art method to get word embedding. Its goal is to discover a set of bases that can represent the words efficiently.

The main structure of this paper is as follows: the models mentioned above and the evaluation methods are introduced in Sect. 4.2; the applications of word embedding are included in Sect. 4.3; conclusion and future work are in Sect. 4.4.

4.2 Word Embedding Approaches and Evaluations

Generally speaking, the goal of word embedding is mapping the words in unlabeled text data to a continuously-valued low dimensional space, in order to capture the internal semantic and syntactic information. In this section, we first introduce the background of text representation. Then, according to the specific methods for generating the mapping, we mainly focus on the Neural Network Language Model and Sparse Coding Approach. The former is further introduced in three parts, depending on the network structures applied in the model. In the end, we provide evaluation approaches for measuring the performance of word embedding models.

4.2.1 Background

One of the widely used approach for expressing the text documents is the Vector Space Model (VSM), where documents are represented as vectors. VSM was originally developed for the SMART information retrieval system (Salton et al. 1997). Some classical VSMs can be found in Deerweste et al. (1990) and Hofmann (2001).

There are various ways of building VSMs. In scenarios of information retrieval where people care more about the textual features that facilitate text categorization, various features selection methods such as document frequency (DF), information gain (IG), mutual information (MI), χ^2-test (CHI-test) and term strength (TS) have different effects on text classification (Yang and Pedersen 1997). These approaches help reduce the dimension of the text data, which could be helpful for the subsequent processing. DF refers to the number of documents that a word appears. IG measures the information loss between presence and absence term in

the document. MI is the ratio between term-document joint probability and the product of their marginal probability. CHI-test applies the sums of squared errors and tries to find out the significant difference between observed word frequency and expected word frequency. TS estimates how likely a term will appear in "closely-related" documents. In information retrieval systems, these methods are useful for transforming the raw text to the vector space, and tremendous improvement has been achieved for information classification. However, it does not work well alone in applications that require both semantic and syntax information of words, since part of the original text information is already lost in feature selection procedures. However, various word embedding models, such as sparse coding, are built based on the VSM.

In Deerweste et al. (1990), Latent Semantic Analysis (LSA) is proposed to index and retrieve the information from text data automatically. It applies SVD on the term-document association data to gain the embedding output. Work in Landauer et al. (1998) and Landauer and Dumais (1997) gives an explanation to the word similarity that derives from the Test of English as a Foreign Language (TOEFL). After that, Landauer (2002) tries to detect the synonym through LSA. Furthermore, Yih et al. (2012) uses LSA to distinguish between synonyms and antonyms in the documents and its extension Multiview LSA (MVLSA) (Rastogi et al. 2015) supports the fusion of arbitrary views of data. However, all of them have to confront the limitation that depends on a single matrix of term-document co-occurrences. Usually the word embedding from those models is not flexible enough, because of the strong reliance on the observed matrix.

Latent Dirichlet Allocation (LDA) (Bengio et al. 2003) is another useful model for feature expression. It assumes that some latent topics are contained in documents. The latent topic T is derived from the conditional distribution $p(w|T)$, i.e. the probability that word w appears in T. Word embedding from traditional LDA models can only capture the topic information but not the syntactic one, which does not fully achieve its goal. Moreover, traditional LDA-based model is only used for topic discovery, but not word representation. But we can get the k-dimensional word embedding by training a k-topic model, the word embedding is from the rows of word-topic matrix and the matrix is filled by $p(w|T)$ values.

Although both of LSA and LDA utilize the statistical information directly for word embedding generation, there are some differences between them. Specifically, LSA is based on matrix factorization and subject to the non-negativity constraint. LDA relies on the word distribution, and it is expressed by the Dirichlet priori distribution which is the conjugate of multinomial distribution.

4.2.2 Neural Network Language Model

The concept of word embedding was first introduced with the Neural Networks Language Model (NNLM) in Xu and Rudnicky (2000) and Bengio et al. (2003). The motivation behind is that words are more likely to share similar meaning if they

are in the same context (Harris 1954). The probability that the word sequence W occurs can be formulated according to the Bayes rule:

$$P(W) = \prod_{t=1}^{N} P(w_t|w_1, \ldots, w_{t-1}) = \prod_{t=1}^{N} P(w_t|h_t) \tag{4.1}$$

where $P(W)$ is the joint distribution of sequence W, and h_t denotes the context words around word w_t. The goal is to evaluate the probability that the word w_t appears given its context information. Because there are a huge number of possible combinations of a word and its context, it is impractical to specify all $P(w_t|h_t)$. In this case, we can use a function Φ to map the context into some equivalent classes, so that we have

$$P(w_t|h_t) = P(w_t|\Phi(h_t)) \tag{4.2}$$

where all of the words are statistically independent.

If we use the n-gram model to construct a table of conditional probability for each word, then the last n-1 words are combined together as the context information:

$$P(w_t|h_t) \approx P(w_t|\tilde{w}_{t-n+1}^{t-1}) \tag{4.3}$$

where n is the number of the words considered, while 1 and 2 are the most commonly used value. Only the successive words occurring before the current word w_t are taken into account.

Most NNLM-based approaches belong to the class of unsupervised models. Networks with various architectures such as Restrict Boltzmann Machine (RBM), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Long-Short Term Memory (LSTM) can be used to build word embedding. Log-Bilinear (LBL) model, SENNA, and Word2Vector are some of the most representative examples of NNLM.

Usually the goal of the NNLM is to maximize or minimize the function of Log-Likelihood, sometimes with additional constraints. Suppose we have a sequence $w_1, w_2, \ldots, w_n$ in the corpus, and we want to maximize the log-likelihood of $P(w_t|\tilde{w}_{t-n+1}^{t-1})$ in the Feed Forward Neural Network (Bengio et al. 2003). Let x be the embedding vector of proceeding words, so that

$$x = [e(w_{t-n+1}), \ldots, e(w_{t-2}), e(w_{t-1})] \tag{4.4}$$

where $e(w)$ represents the embedding of word w. In Feed Forward Neural Network (without the direct edge in the structure) with one hidden layer, the layer function can be expressed as:

$$y = b + U(tanh(d + Wx)) \tag{4.5}$$

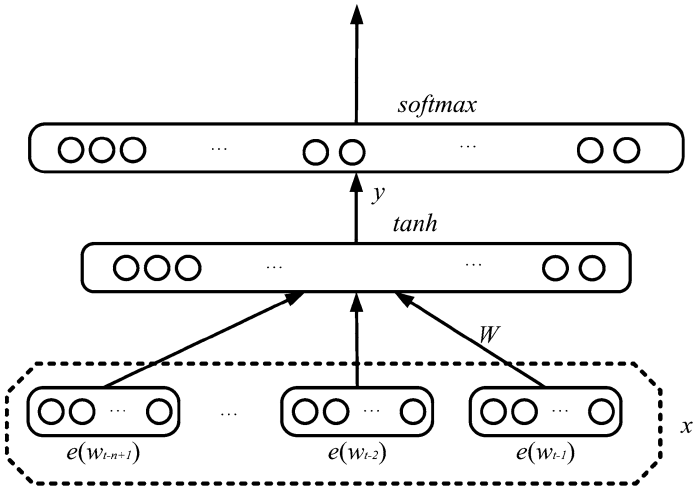

Fig. 4.1 The basic structure of the Feed Forward Neural Network

where U is the transformation matrix, W is the weight matrix of hidden layer, b and d are bias vectors. Finally, y is fed into a softmax layer to obtain the probability of the target word, and the basic structure of this model is shown in Fig. 4.1. The parameters θ in this model are (b, U, d, W). However, since we need to explicitly normalize (e.g. use softmax) all of the values in the vocabulary when computing the probability for the next word, it will be very costly for training and testing (Morin and Bengio 2005). After estimating the conditional probability in Eq. (4.3), the total probability $P(w_{t-n+1}, .., w_{t-1}, w_t)$ can be obtained directly by locating the mark of the phrase that is constructed by the words that appears in the same window (Collobert et al. 2011). The resultant probability value quantifies the normality if such a sentence appears in the natural language (more details can be found in Sect. 4.2.2.3).

Instead of using matrix multiplication on the final layer (Bengio et al. 2003; Mnih and Hinton 2007) applies the hierarchical structure (see Sect. 4.2.2.4) to reduce the computational complexity. It constructs the Log-BiLinear (LBL) model by adding bilinear interaction between word vectors and hidden variables on the basis of the energy function (see Sect. 4.2.2.1).

Recently, Word2Vector proposed by Mikolov et al. (2013) which contains Skip-Gram and (Continuous Bag-of-Words) CBOW these two frameworks is very popular in NLP tasks. It can be seen as a two-layer Neural Network Language Model. Furthermore, applying Noise Contrastive Estimation (NCE) increases its efficiency. However, we can also add the priori information into this model, Liu et al. (2015) extends Skip-Gram by treating topical information as important priori

knowledge for training word embedding and proposes the topical word embedding (TWE) model, where text words and their affiliated topic (context) derived from LDA are combined to obtain the embedding.

In the subsections below, we will introduce some basic architectures of NNLM for word embedding.

4.2.2.1 Restricted Boltzmann Machine (RBM)

The ability of capturing the latent features among words usually depends on the model structure. Boltzmann Machine (BM) originates from the log-linear Markov Random Field (MRF) and its energy function is linearly related to free parameters. According to the statistical dynamics, energy function is useful in word embedding generation (Mnih and Hinton 2007). Restricted Boltzmann Machine (RBM), which prunes the visible-visible and hidden-hidden connections, is a simplified version of BM. A diagram of RBM is given in Fig. 4.2.

The main components of the energy function in RBM include binary variables (hidden unites h and visible unites v), weights $W = (w_{i,j})$ that establish connections between h_j and v_i, biases a_i for the visible units and b_j for the hidden ones. Specifically, the energy function is formulated as follows,

$$E(v, h) = -v^T Wh - a^T v - b^T h \tag{4.6}$$

During the generation process word embedding, the energy function is used as the probability distribution as shown in Eq. (4.7).

$$P(v, h) = e^{-E(v,h)}/Z \tag{4.7}$$

where Z is a partition function defined as the sum of $e^{-E(h,v)}$.

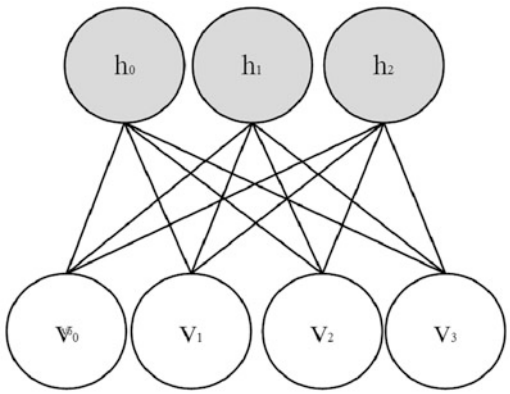

Fig. 4.2 The basic structure of the RBM

Mnih and Hinton (2007) proposes three models on the basis of RBM by adding connections to the previous status. Starting from the undirected graphical model, they try to estimate the latent conditional distribution of words. To compute the probability of the next word in a sequence, the energy function is defined as follows:

$$E(w_t, h; w_{t-n+1:t-1}) = -(\sum_{i=t-n+1}^{t-1} v_i^T R W_i)h - b_h^T h - b_t^T R^T v_n - b_v^T v_n \tag{4.8}$$

where $v_i^T R$ denotes the vector that represents word w_i from the word dictionary R, W_i in hidden units denotes the similarity between two words, and b_h, b_v, b_r are the biases for the hidden units, words and word features respectively.

The disadvantage of this class of models is that it is costly to estimate massive parameters during the training process. To reduce the number of parameters, Mnih and Hinton (2007) extends the factored RBM language model by adding connections C among words. It also removes the hidden variables and directly applies the stochastic binary variables. The modified energy function is as below:

$$E(w_t; w_{t-n+1:t-1}) = -(\sum_{i=t-n+1}^{t-1} v_i^T R C_i)R^T v_n - b_r^T R^T v_n - b_v^T v_n \tag{4.9}$$

Here C_i denotes correlation between word vectors w_i and w_t, b_r and b_v denotes the word biases. This function quantifies the bilinear interaction between words, which is also the reason why models of this kind are called Log-BiLinear Language (LBL) models. In Eq. (4.9), entries of vector $h = \sum_{i=t-n+1}^{t-1} v_i^T R C_i$ correspond to the nodes in the hidden layer, and $y_i = hR^T v_n$ represents the set of nodes in the output layer. We can see that the syntax information is contained in the hidden layer, for C_i here can be seen as the contribution of word i to current word n, which is like the cosine similarity between two words.

4.2.2.2 Recurrent and Recursive Neural Network

To reduce the number of parameters, we could unify the layer functions with some repetitive parameters. By treating the text as sequential data, Mikolov et al. (2010) proposes a language model based on the Recurrent Neural Network, the structure of which is shown in Fig. 4.3. The model has a circular architecture. At time t, word embedding $w(t)$ is generated out of the first layer. Then, it is transferred together with the output from the context layer at time $t-1$ as the new input $x(t)$ to the context layer, which is formulated as follows:

$$x(t) = w(t) + s(t-1) \tag{4.10}$$

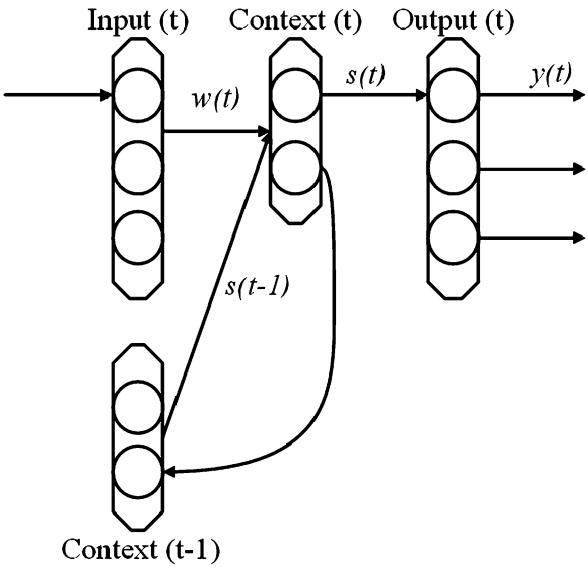

Fig. 4.3 The structure of the recurrent neural network

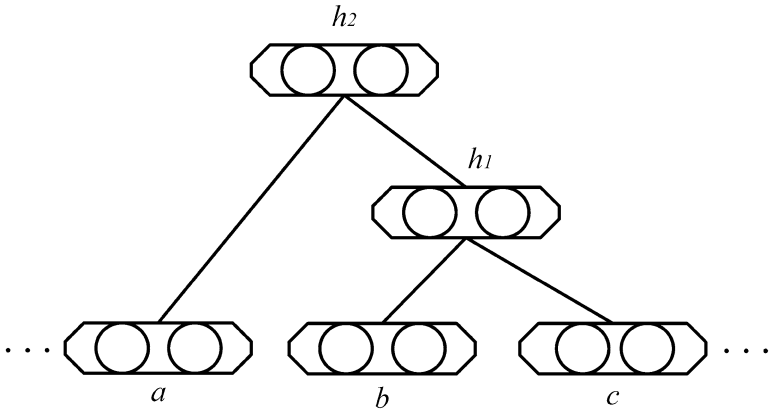

Fig. 4.4 The structure of the Recursive Neural Network

In each cycle, the outcomes of context layer and output layer are denoted as $s(t)$ and $y(t)$ respectively. Inspired by Bengio et al. (2003), the output layer also applies the softmax function.

Unlike Recurrent Neural Network, Recursive Neural Network (Goller and Kuchler 1996) is a linear chain in space as shown in Fig. 4.4. To fully analyze the sentence structure, the process iterates from two word vectors in leaf nodes b, c to merge into a hidden output h_1, which will continue to concatenate another input word vector a as the next input. The computing order is determined either by the sentence structure or the graph structure. Not only could hidden layers capture the context information, they could also involve the syntax information. Therefore,

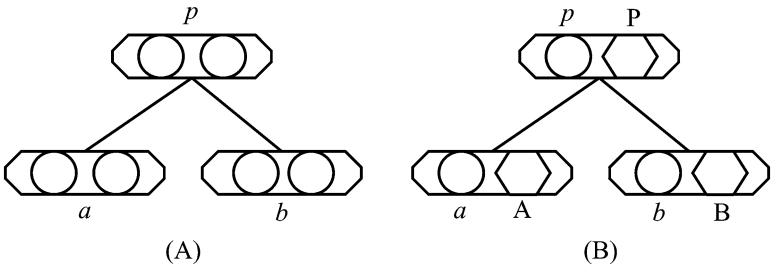

Fig. 4.5 (**A**) The structure of the Recursive Neural Network model where each node represents a vector and all the word vectors are in the leaf nodes. (**B**) The structure of the MV-RNN model where each node consists of a vector and a matrix

Recursive Neural Network can capture more information than the context-based model, which is desirable for NLP tasks.

For the sequential text data in Recursive Neural Network, Socher et al. (2011, 2013, 2012) parses the sentences into tree structures through the tree bank models (Klein and Manning 2003; Antony et al. 2010). The model proposed by Socher et al. (2011) employs the Recursive Neural Network, and its structure is shown in Fig. 4.5A,

$$p = f(W \begin{bmatrix} a \\ b \end{bmatrix}) \tag{4.11}$$

where $a, b \in R^{d\times 1}$ are the word vectors, parent node $p \in R^{d\times 1}$ is the hidden output, and $f = tanh$ adds element-wise nonlinearity to the model. $W \in R^{d\times 2d}$ is the parameter to learn.

To incorporate more information in each node, work in (Socher et al. 2012) proposes the model of Matrix-Vector Recursive Neural Network (MV-RNN), for which the parsing-tree structure is shown in Fig. 4.5B. Different from that in Fig. 4.5A, each node in this parsing tree includes a vector and a matrix. The vector represents the word vector in leaf nodes and phrase in non-leaf ones. The matrix is applied to neighboring vectors, and it is initialized by the identity matrix I plus a Gaussian noise. The vectors a, b, p capture the inherent meaning transferred in the parsing tree, and the matrixes A, B, P are to capture the changes of the meaning for the neighboring words or phrases. Apparently, the drawback of this model is the large number of parameters to be estimated. So in the work of Socher et al. (2013), the model of Recursive Neural Tensor Network (RNTN) uses the tensor-based composition function to avoid this problem.

In comparison, the main differences between recurrent and recursive models are the computing order and the network structure. The structure of the former has a closed loop dealing with the time sequence, while the latter has an open loop tackling the spatial sequence.

4.2.2.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) consists of several feature extraction layers, which is inspired by biological process (Matsugu et al. 2003). It has already been successfully applied in many image recognition problems. The main advantage of CNN is that it does not depend on the prior knowledge or human efforts when doing feature selection.

CNN can also be applied to extract latent features from text data (Collobert and Weston 2008; Collobert et al. 2011). As shown in Fig. 4.6, three different convolutional kernels, in three different colors, select different features from word embedding separately. The feature vector is the combination of the max values from the rows of the selected features.

In the models above, supervised learning is applied to train the whole neural network, and the convolutional layer here is to model the latent feature of the initialized word embedding. Since supervised training method is applied here, it is able to generate word embedding suitable for the special task. Though the word embedding is usually a by-product in this case, it still has a sound performance in capturing semantic and syntactic information.

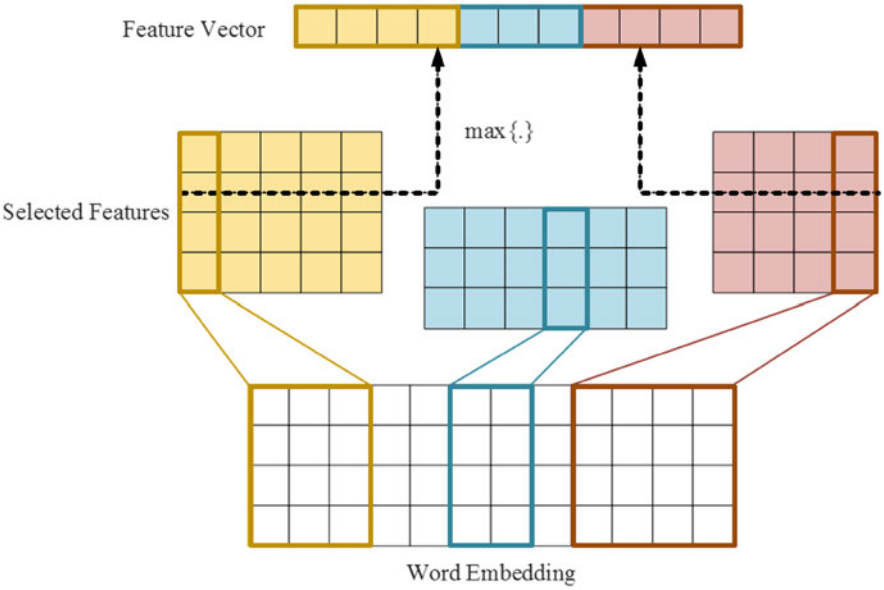

Fig. 4.6 The structure of CNN for feature extraction by using convolutional kernels

4.2.2.4 Hierarchical Neural Language Model (HNLM)

There are many tricks to speed up the process of training NNLM, such as short list, hash table for the word and stochastic gradient descent (Bengio et al. 2003). But when facing with datasets of large scale, further improvement in model efficiency is still needed.

Transferring the knowledge from a small portion of observed data examples to other cases can save a lot of computational efforts, thus speeding up the learning process. However, it is a difficult task for neural language models which involve computing the joint probability of certain word combinations. This is because the possible combinations of words from a vocabulary is immensely larger than the ones from all potential texts. So it is necessary to look for some priori information before searching all possible combinations. Based on human knowledge or statistical information, we can cluster all the words into several classes where some similar statistical properties are shared by the words in each class. Some hierarchical models are constructed based on this idea (Mnih and Hinton 2008; Morin and Bengio 2005).

Hierarchical structures has a big influence on the algorithm complexity. For example, Mnih and Hinton (2008) and Morin and Bengio (2005) reduces the complexity of language models by clustering words into a binary tree. Specifically, given a dictionary V containing $|V|$ words, the speed up will be $O(|V|/log|V|)$ (Morin and Bengio 2005) if the tree is binary.

To determine the hierarchical structure, hierarchical word classes (lexical word categories in a narrow sense) are needed in most cases (Rijkhoff and Jan 2007). The way of constructing word classes is sensitive to the word distribution. Word classes can be built based on prior (expert) knowledge (Fellbaum 1998) or through data-driven methods (Sun et al. 2012), but they could also be automatically generated in accordance with the usage statistics (Mnih and Hinton 2008; McMahon and Smith 1996). The prior knowledge is accurate but limited by the application range, since there are lots of problems where expert categories are unavailable. The data-driven methods can boost a high level of accuracy but it still needs a seed corpus which is manually tagged. Universality is the major advantage of the automatic generation method. It could be applied to any natural language scenarios without the expert knowledge, though it will be more complex to construct.

By utilizing the hierarchical structure, lots of language models have been proposed (Morin and Bengio 2005; Mnih and Hinton 2008; Yogatama et al. 2014a; Djuric et al. 2015). Morin and Bengio (2005) introduces the hierarchical decomposition bases on the word classes that extract from the WordNet. Similar with Bengio et al. (2003), it uses the layer function to extract the latent features, the process is shown in Eq. (4.12).

$$P(d_i = 1|q_i, w_{t-n+1:t-1}) = sigmoid(U(tanh(d + Wx)_{q_i}) + b_i) \tag{4.12}$$

Here x is the concatenation of the context word [same as Eq. (4.4)], b_i is the biases vector, U, W, b_i, x plays the same roles as in Eq. (4.5). The disadvantage is the procedure of tree construction which has to combine the manual and data-driving

processing (Morin and Bengio 2005) together. Hierarchical Log BiLinear (HLBL) model which is proposed by Mnih and Hinton (2008) overcomes the disadvantage by using a boosting method to generate the tree automatically. The binary tree with words as leaves consists of two components: the words in the leaves which can be represented by a sequential binary code uniquely from top to down, as well as the probability for decision making at each node. Each non-leaf node in the tree also has an associated vector q which is used for discrimination. The probability of the predicted word w in HLBL is formulated as follows

$$P(w_t = w|w_{t-n+1:t-1}) = \prod_i P(d_i|q_i, w_{t-n+1:t-1}) \tag{4.13}$$

where d_i is t^{th} digit in the binary code sequence for word w, and q_i is the feature vector for the i^{th} node in the path to word w from the root. In each non-leaf node, the probability of the decision is given by

$$P(d_i = 1|q_i, w_{t-n+1:t-1}) = \tau((\sum_{j=t-n+1}^{t-1} C_j w_j)\dot{q}_i + b_i) \tag{4.14}$$

where $\tau(x)$ is the logistic function for decision making, C_j is the weight matrix corresponding to the context word w_j, and b_i is the bias to capture the context-independent tendency to one of its children when leaving this node.

There are both advantages and disadvantages of the hierarchical structure. Many models benefit from the reduction of computation complexity, while the consuming time on the structure building is the main drawback.

4.2.3 Sparse Coding Approach

Sparse Coding is an unsupervised model that learns a set of over-complete bases to represent data efficiently. It generates base vectors $\{\phi_i\}$ such that the input vector $x \in R^n$ can be represented by a linear combination of them $x = \sum_{i=1}^{k} \alpha_i \phi_i$, where k is the number of base vectors and $k > n$. Although there are many techniques such as Principal Component Analysis (PCA) that helps learn a complete set of basis vectors efficiently, Sparse Coding aims to use the least bases to represent the input x. The over-complete base vectors are able to capture the structures and patterns inherent in the data, so the nature characteristic of the input can be seized through Sparse Coding.

To be specific, sparse Coding constructs an over-complete dictionary $D \in R^{L \times K}$ of basis vectors, together with a code matrix $A \in R^{K \times V}$, to represent V words in C contents $X \in R^{L \times V}$ by minimizing such function as follows:

$$\arg\min_{D,A} \|X - DA\|_2^2 + \lambda \Omega(A) \tag{4.15}$$

where λ is a regularization hyperparameter and Ω is the regularizer. It applies the squared loss for the reconstructed error, but loss functions like L_1-regularized quadratic function can also be an efficient alternative for this problem (Schökopf et al. 2007).

Plenty of algorithms have been proposed to extend the sparse coding framework starting from the general loss function above. If we apply a non-negativity constraint on A (i.e. $A_{i,j} \geq 0$), the problem is turned into the Non-Negative Sparse Embedding (NNSE). Besides adding constraints on A, if we also requires that all entries in D are bigger than 0, then the problem can be transformed into the Non-Negative Sparse Coding (NNSC). NNSC is a matrix factorization technique previously studied in machine learning communities (Hoyer 2002).

For $\Omega(A)$, Yogatama et al. (2014b) designs a forest-structured regularizer that enables the mixture use of the dimension. The structure of this model is in Fig. 4.7, there are seven variables in the tree which describes the order that variables 'enter the model'. In this work, it sets the rule that a node may take a nonzero value only if its ancestors do. For example, nodes 3 and 4 may only be nonzero if nodes 1 and 2 are also nonzero. So the regularizer for A shows in Eq. (4.16).

$$\Omega(a_v) = \sum_{i=1}^{V} \left\| \langle a_{v,i}, a_{v,Descendants(i)} \rangle \right\|_2 \tag{4.16}$$

where a_v is the vth column of A.

Faruqui et al. (2015) proposes two methods by adding restriction on the dictionary D:

$$\arg\min_{D,A} \|X - DA\|_2^2 + \lambda\Omega(A) + \tau \|D\|_2^2 \tag{4.17}$$

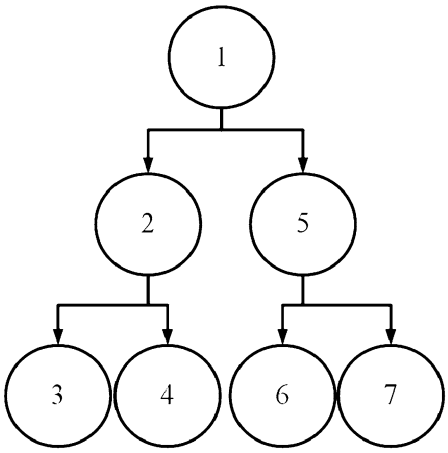

Fig. 4.7 An example of a regularization forest that governs the order in which variables enter the model

where τ is the regularization hyperparameter. The first method applies l_1 penalty on A, so that the Function 4.17 can be broken into the loss for each word vector, which makes it possible for parallel optimization.

$$\arg\min_{D,A} \sum_{i=1}^{V} \|x_i - Da_i\|_2^2 + \lambda \|a_i\|_1 + \tau \|D\|_2^2 \tag{4.18}$$

Here x_i and a_i are the ith column vector of matrix X and A respectively. The second method is to add non-negativity constraints on variables so that:

$$\arg\min_{D \in R_{\geq 0}^{L \times K}, A \in R_{\geq 0}^{K \times V}} \sum_{i=1}^{V} \|x_i - Da_i\|_2^2 + \lambda \|a_i\|_1 + \tau \|D\|_2^2 \tag{4.19}$$

Apart from the work based on Sparse Coding, Sun et al. (2016) adds the l_1 regularizer into Word2Vector. The challenge in this work lies in the optimization method, for stochastic gradient descent (SGD) cannot produce sparse solution directly with l_1 regularizer in online training. The method of Regularized Dual Averaging (RDA) proposed by Lin (2009) keeps track of online average subgradients at each update, and optimizes l_1 regularized loss function based on Continuous Bag-of-Words (CBOW) Model (Mikolov et al. 2013) in online learning.

4.2.4 Evaluations of Word Embedding

Measurements for word embedding usually depend on the specific applications. Some representative examples include perplexity, analogy precision and sentiment classification precision.

Perplexity is an evaluation method which originates from information theory. It measures how well a distribution or a model can predict the sample. Bengio et al. (2003) uses the perplexity of the corpus as the target, where the lower of the perplexity value is, the higher quality the word embedding conveys since the information is more specific. Word or phrase analogy analysis is another important assessment way. Work in (Mikolov et al. 2013) designs the precision of semantic and syntactic prediction as the standard measurement to evaluate the quality of word embedding. It is applied in phrase and word level independently. All the assessment methods mentioned above are from the view of linguistic phenomenon that the distance between two words in vector space reflects the correlation between them.

Besides the measurements of linguistic phenomenon, a lot of work evaluates word embedding by using the task where they are applied. For example, if the word embedding is used in sentiment classification, then the precision of the classification can be used for evaluation. If it is applied in machine translation, then the precision

of the translation is the one that matters. This rule also holds for other tasks like POS tagging, named entity recognize, textual entailment, and so on.

There is no best measurement method for all scenarios. The most useful way is to combine it with the target of the task to achieve a more reasonable evaluation result.

4.3 Word Embedding Applications

There are many applications for word embedding, especially in NLP tasks where word embedding is fed as input data or the features of the text data directly. In this section, we will discuss how word embedding can be applied in the scenarios such as semantic analysis, syntax analysis, idiomaticity analysis, Part Of the Speech (POS) tagging, sentiment analysis, named entity recognition, textual entailment as well as machine translation.

The goal of syntax analysis is to extract the syntactic structure from sentences. Recent works (Socher et al. 2011; Huang et al. 2012; Collobert and Weston 2008; Mikolov et al. 2013) have taken the syntax information into consideration to obtain word embedding, and the result in Andreas and Dan (2014) shows that word embedding can entail the syntactic information directly. A fundamental problem to syntax analysis is part of the speech tagging, which is about labeling the words to different categories (e.g., plural, noun, adverb). It requires the knowledge of definitions and contextual information. Part of the speech tagging is a word-level NLP task. Collobert and Weston (2008) and Collobert et al. (2011) apply word embedding for POS tagging and achieve state-of-the-art results. In some scenarios, in order to figure out the syntax of text, we need to first recognize the named entities in it. Named entity recognition aims to find out the names of persons, organizations, time expressions, monetary values, locations and so on. Works in Collobert and Weston (2008), Collobert et al. (2011), Zou et al. (2013), Luo et al. (2014), Pennington et al. (2014) show the expressiveness of word embedding in these applications.

As a subsequent task, semantic analysis (Goddard 2011) relates the syntactic structures of words to their language-independent meanings.[1] In other words, it reveals words that are correlated with each other. For example, the vector ("Madrid" − "Spain" + "France") is close to ("Paris"). Previous approaches (Scott et al. 1999; Yih et al. 2012) mainly use the statistical information of the text. The works in Mikolov et al. (2013), Socher et al. (2011), Huang et al. (2012), Collobert and Weston (2008) apply the analogy precision to measure the quality of word embedding (see in Sect. 4.2.4). It is shown that word embedding can manifest the semantic information of words in the text. The semantics of each word can help us understand the combinatory meaning of several words. The phenomenon of multi-words expression idiomaticity is common in nature language, and it is difficult to be

[1] https://en.wikipedia.org/wiki/Semantic_analysis_(linguistics).

inferred. For example, the meaning of "ivory tower" could not be inferred from the separate words "ivory" and "tower" directly. Different from semantic analysis and syntax analysis, idiomaticity analysis (Salehi et al. 2015) requires the prediction on the compositionality of the multi-words expression. Once we figure out the syntax and semantics of words, we can do sentiment analysis whose goal is to extract the opinion, sentiment as well as subjectivity from the text. Word embedding can act as the text features in the sentiment classifier (Socher et al. 2012; Dickinson and Hu 2015).

In some NLP problems, we need to deal with more than one type of language corpus. Textual entailment (Saurf and Pustejovsky 2007) is such a task that involves various knowledge sources. It attempts to deduce the meaning of the new text referred from other known text sources. Works in Bjerva et al. (2014) and Zhao et al. (2015) apply word embedding into the score computing in accordance with the clustering procedure. Machine translation is another important field in NLP, which tries to substitute the texts in one language for the corresponding ones in another language. Statistical information is widely used in some previous works (Ueffing et al. 2007). Neural network models are also a class of important approaches. Bahdanau et al. (2014) builds the end-to-end model that is based on the encoder-decoder framework, while it is still common for machine translation to use word embedding as the word expression (Mikolov et al. 2013; Hill et al. 2014). Furthermore, Zou et al. (2013) handles the translation task using bilingual embedding which is shown to be a more efficient method.

According to the type of roles that it acts in the applications, word embedding could be used as the selected feature or in the raw digital format. If word embedding is used as the selected feature, it is usually used for the higher level NLP tasks, like sentiment classification, topic discovery, word clustering. In linguistics tasks, word embedding is usually used in its raw digital format, like semantic analysis, syntax analysis and idiomaticity analysis. Based on the type of machine learning problems involved, applications of word embedding could be divided into regression task, clustering task and classification task. Works in Zhao et al. (2015) prove that word embedding could make improvements for regression and classification problems which attempt to find out the pattern of the input data and make prediction after the fitting. Tasks like semantic analysis, idiomaticity analysis and machine translation belong to this category. Luo et al. (2014) tries to find out the similarity between short texts by applying a regression step in word embedding, where the resultant model could be used for searching, query suggestion, image finding, and so on. Word embedding is also commonly used in classification tasks like sentiment analysis and textual entailment (Amir et al. 2015). POS tagging and named entity recognition discussed above belong to clustering problems.

Besides word embedding, phrase embedding and document embedding are some other choices for expressing the words in the text. Phrase embedding vectorizes the phrases for higher level tasks, such as web document management (Sharma and Raman 2003), paraphrase identification (Yin and Schütze 2016) and machine translation (Zou et al. 2013). Document embedding treats documents as basic units. It can be learned from documents directly (Huang et al. 2013) or aggregated

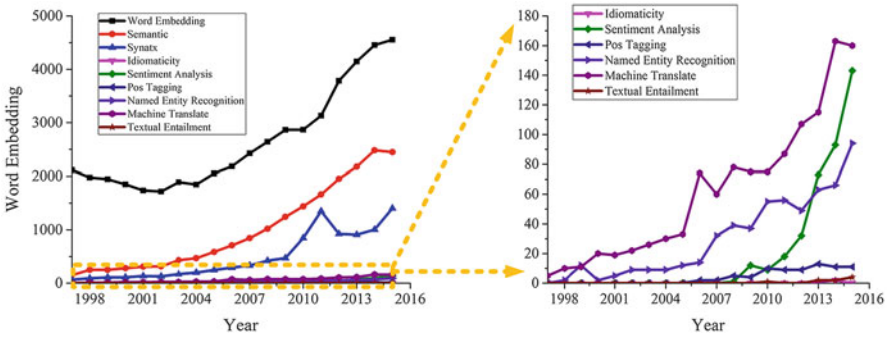

Fig. 4.8 The citation numbers of the topics in each year

by word embedding (Lin and He 2009; Zhou et al. 2015). Similar to phrase embedding, document embedding can also be applied in sentiment analysis and machine translation.

Figure 4.8 summarizes the application distribution of word embedding in different NLP tasks.[2] We can see that word embedding stably gains its popularity starting from 2004, as reflected though the rising curve. One of the most common applications is semantic analysis, in which nearly half of the works of word embedding are involved. Then it comes to the syntax analysis whose popularity dramatically increases between 2009 and 2011. Compared with syntax and semantic analysis, although other applications account for a much less proportion of work, domains like machine translation, names entity recognition and sentiment analysis receive dramatically increasing attention since 2010.

4.4 Conclusion and Future Work

In conclusion, word embedding is a powerful tool for many NLP tasks, especially the ones that require original input as the text features. There are various types of models for building word embedding, and each of them has its own advantages and disadvantages. Word embedding can be regarded as textual features, so that it can be counted as a preprocessing step in more advanced NLP tasks. Not only can it be fed into classifiers, but it can be also used for clustering and regression problems. Regarding the level that embedding represents, word embedding is a fine-grit representation compared with phrase embedding and document embedding.

Word embedding is an attractive research topic worth of further exploration. First, to enrich the information contained in word embedding, we can try to involve various prior knowledge such as synonymy relations between words, domain

[2]The citation numbers are from http://www.webofscience.com.

specific information, sentiment information and topical information. The resultant word embedding generated towards this direction will be more expressive. Then, besides using words to generate embedding, we may want to explore how character-level terms can affect the output. This is due to the reason that words themselves are made up of character-level elements. Such morphological analysis matches the way of how people perceive and create words, thus can help deal with the occurrences of new words. Moreover, as data volume is fast accumulating nowadays, it is necessary to develop techniques capable of efficiently process huge amount of text data. Since text data may arrive in streams, word embedding models that incorporate the idea of online learning are more desirable in this scenario. When the new chunk of data is available, we do not need to learn a new model using the entire data corpus. Instead, we only need to update the original model to fit the new data.

References

Amir, S., Astudillo, R., Ling, W., Martins, B., Silva, M. J., & Trancoso, I. (2015). INESC-ID: A regression model for large scale twitter sentiment lexicon induction. In *International Workshop on Semantic Evaluation*.

Andreas, J., & Dan, K. (2014). How much do word embeddings encode about syntax? In *Meeting of the Association for Computational Linguistics* (pp. 822–827).

Antony, P. J., Warrier, N. J., & Soman, K. P. (2010). Penn treebank. *International Journal of Computer Applications, 7*(8), 14–21.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. Eprint arxiv.

Bengio, Y., Schwenk, H., Senécal, J. S., Morin, F., & Gauvain, J. L. (2003). A neural probabilistic language model. *Journal of Machine Learning Research, 3*(6), 1137–1155.

Bjerva, J., Bos, J., van der Goot, R., & Nissim, M. (2014). The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *SemEval-2014 Workshop*.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. In *International Conference, Helsinki, Finland, June* (pp. 160–167).

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research, 12*(1), 2493–2537.

Deerweste, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Richard (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science, 41*, 391–407.

Dickinson, B., & Hu, W. (2015). Sentiment analysis of investor opinions on twitter. *Social Networking, 04*(3), 62–71.

Djuric, N., Wu, H., Radosavljevic, V., Grbovic, M., & Bhamidipati, N. (2015). Hierarchical neural language models for joint representation of streaming documents and their content. In *WWW*.

Faruqui, M., Tsvetkov, Y., Yogatama, D., Dyer, C., & Smith, N. (2015). Sparse overcomplete word vector representations. Preprint, arXiv:1506.02004.

Fellbaum, C. (1998). *WordNet*. Wiley Online Library.

Goddard, C. (2011). *Semantic analysis: A practical introduction*. Oxford: Oxford University Press.

Goller, C., & Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks* (Vol. 1, pp. 347–352).

Harris, Z. S. (1954). Distributional structure. *Synthese Language Library, 10*(2–3), 146–162.

Hill, F., Cho, K., Jean, S., Devin, C., & Bengio, Y. (2014). Embedding word similarity with neural machine translation. Eprint arXiv.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of CogSci*.

Hofmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning, 42*(1–2), 177–196.

Hoyer, P. O. (2002). Non-negative sparse coding. In *IEEE Workshop on Neural Networks for Signal Processing* (pp. 557–565).

Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *Meeting of the Association for Computational Linguistics: Long Papers* (pp. 873–882).

Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., & Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13 (pp. 2333–2338). New York, NY: ACM.

Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. In *Meeting on Association for Computational Linguistics* (pp. 423–430).

Lai, S., Liu, K., Xu, L., & Zhao, J. (2015). How to generate a good word embedding? *Credit Union Times*, III(2).

Landauer, T. K. (2002). On the computational basis of learning and cognition: Arguments from lsa. *Psychology of Learning & Motivation, 41*(41), 43–84.

Landauer, T. K., & Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review, 104*(2), 211–240.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes, 25*(2), 259–284.

Lin, C., & He, Y. (2009). Joint sentiment/topic model for sentiment analysis. In *ACM Conference on Information & Knowledge Management* (pp. 375–384).

Lin, X. (2009). Dual averaging methods for regularized stochastic learning and online optimization. In *Conference on Neural Information Processing Systems 2009* (pp. 2543–2596).

Liu, Y., Liu, Z., Chua, T. S., & Sun, M. (2015). Topical word embeddings. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Luo, Y., Tang, J., Yan, J., Xu, C., & Chen, Z. (2014). Pre-trained multi-view word embedding using two-side neural network. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Matsugu, M., Mori, K., Mitari, Y., & Kaneda, Y. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks, 16*(5–6), 555–559.

McMahon, J. G., & Smith, F. J. (1996). Improving statistical language model performance with automatically generated word hierarchies. *Computational Linguistics, 22*(2), 217–247.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH 2010, Conference of the International Speech Communication Association*, Makuhari, Chiba, Japan, September (pp. 1045–1048).

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

Mnih, A., & Hinton, G. E. (2008). A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, Vancouver, British Columbia, Canada, December 8–11, 2008 (pp. 1081–1088).

Morin, F., & Bengio, Y. (2005). Hierarchical probabilistic neural network language model. *Aistats* (Vol. 5, pp. 246–252). Citeseer.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Rastogi, P., Van Durme, B., & Arora, R. (2015). Multiview LSA: Representation learning via generalized CCA. In *Conference of the North American chapter of the association for computational linguistics: Human language technologies, NAACL-HLT'15* (pp. 556–566).

Rijkhoff, & Jan (2007). Word classes. *Language & Linguistics Compass, 1*(6), 709–726.

Salehi, B., Cook, P., & Baldwin, T. (2015). A word embedding approach to predicting the compositionality of multiword expressions. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Salton, G., Wong, A., & Yang, C. S. (1997). *A vector space model for automatic indexing*. San Francisco: Morgan Kaufmann Publishers Inc.

Saurf, R., & Pustejovsky, J. (2007). Determining modality and factuality for text entailment. In *International Conference on Semantic Computing* (pp. 509–516).

Schökopf, B., Platt, J., & Hofmann, T. (2007). Efficient sparse coding algorithms. In *NIPS* (pp. 801–808).

Scott, D., Dumais, S. T., Furnas, G. W., Lauer, T. K., & Richard, H. (1999). Indexing by latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 391–407).

Sharma, R., & Raman, S. (2003). Phrase-based text representation for managing the web documents. In *International Conference on Information Technology: Coding and Computing* (pp. 165–169).

Shazeer, N., Doherty, R., Evans, C., & Waterson, C. (2016). Swivel: Improving embeddings by noticing what's missing. Preprint, arXiv:1602.02215.

Socher, R., Huval, B., Manning, C. D., & Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (pp. 1201–1211).

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., & Manning, C. D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27–31 July 2011, John Mcintyre Conference Centre, Edinburgh, A Meeting of SIGDAT, A Special Interest Group of the ACL* (pp. 151–161).

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods on Natural Language Processing*.

Sun, F., Guo, J., Lan, Y., Xu, J., & Cheng, X. (2015). Learning word representations by jointly modeling syntagmatic and paradigmatic relations. In *AAAI*.

Sun, F., Guo, J., Lan, Y., Xu, J., & Cheng, X. (2016). Sparse word embeddings using l1 regularized online learning. In *International Joint Conference on Artificial Intelligence*.

Sun, S., Liu, H., Lin, H., & Abraham, A. (2012). Twitter part-of-speech tagging using pre-classification hidden Markov model. In *IEEE International Conference on Systems, Man, and Cybernetics* (pp. 1118–1123).

Ueffing, N., Haffari, G., & Sarkar, A. (2007). Transductive learning for statistical machine translation. In *ACL 2007, Proceedings of the Meeting of the Association for Computational Linguistics*, June 23–30, 2007, Prague (pp. 25–32).

Xu, W., & Rudnicky, A. (2000). Can artificial neural networks learn language models? In *International Conference on Statistical Language Processing* (pp. 202–205).

Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Fourteenth International Conference on Machine Learning* (pp. 412–420).

Yih, W.-T., Zweig, G., & Platt, J. C. (2012). Polarity inducing latent semantic analysis. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12 (pp. 1212–1222). Stroudsburg, PA: Association for Computational Linguistics.

Yin, W., & Schütze, H. (2016). Discriminative phrase embedding for paraphrase identification. Preprint, arXiv:1604.00503.

Yogatama, D., Faruqui, M., Dyer, C., & Smith, N. A. (2014a). Learning word representations with hierarchical sparse coding. Eprint arXiv.

Yogatama, D., Faruqui, M., Dyer, C., & Smith, N. A. (2014b). Learning word representations with hierarchical sparse coding. Eprint arXiv.

Zhao, J., Lan, M., Niu, Z. Y., & Lu, Y. (2015). Integrating word embeddings and traditional NLP features to measure textual entailment and semantic relatedness of sentence pairs. In *International Joint Conference on Neural Networks* (pp. 32–35).

Zhou, C., Sun, C., Liu, Z., & Lau, F. (2015). Category enhanced word embedding. Preprint, arXiv:1511.08629.

Zou, W. Y., Socher, R., Cer, D. M., & Manning, C. D. (2013). Bilingual word embeddings for phrase-based machine translation. In *EMNLP* (pp. 1393–1398).