Learning Sparse Overcomplete Word Vectors Without Intermediate Dense Representations

Yunchuan Chen[1,2], Ge Li[1,3(✉)], and Zhi Jin[1,3(✉)]

[1] Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China
`chenyunchuan11@mails.ucas.ac.cn`, `{lige,zhijin}@pku.edu.cn`
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Institute of Software, Peking University, Beijing, China

Abstract. Dense word representation models have attracted a lot of interest for their promising performances in various natural language processing (NLP) tasks. However, dense word vectors are uninterpretable, inseparable, and time and space consuming. We propose a model to learn sparse word representations directly from the plain text, rather than most existing methods that learn sparse vectors from intermediate dense word embeddings. Additionally, we design an efficient algorithm based on noise-contrastive estimation (NCE) to train the model. Moreover, a clustering-based adaptive updating scheme for noise distributions is introduced for effective learning when NCE is applied. Experimental results show that the resulting sparse word vectors are comparable to dense vectors on the word analogy tasks. Our models outperform dense word vectors on the word similarity tasks. The sparse word vectors are much more interpretable, according to the sparse vector visualization and the word intruder identification experiments.

1 Introduction

Word representation learning is aimed at associating each word with a syntactically and semantically rich feature vector. The learned word vectors could serve as input features for higher-level algorithms in NLP applications. Based on the distributional hypothesis [11], a variety of methods have been proposed in the NLP community, such as clustering-based methods [1], matrix-based methods [13,15], and neural network-based methods [3,17]. Recently, neural network-based methods have dominated word representation learning because of their effectiveness in a variety of natural language processing (NLP) tasks, such as part-of-speech tagging, semantic role labeling [3], parsing [26], sentiment analysis [27], language modeling [28], paraphrase detection [5] and dialogue analysis [12]. Neural network-based methods often represent words with dense, real-valued vectors.

However, dense representations are often criticized on their interpretability, separability, and complexity aspects. For neural network induced word vectors,

G. Li et al. (Eds.): KSEM 2017, LNAI 10412, pp. 3–15, 2017.
DOI: 10.1007/978-3-319-63558-3_1

we do not know what feature is represented by each dimension, which corresponds to an implicit feature. There are complex dependencies between these underlying implicit features and human interpretable features such as *noun*, *adjective*, *plural* or *singular*, etc. It is therefore difficult to understand how a word vector-based computational model works. In addition, dense word vectors go against some widely believed properties a good high-level representation should have. For example, it is thought features should be represented in a separable way, or related to each other through simple dependencies [8], but dense word vectors are generally not separable. In addition, it usually results in high time or space complexity models when applied in downstream NLP tasks.

Sparse representation is considered as a potential choice for interpretable word representations and can be used to design time and space efficient algorithms for downstream NLP tasks. In the image, speech, or signal processing field, sparse overcomplete representations have been widely used as a way to improve separability and interpretability [4,21,25], and to increase stability in the presence of noise [4]. In NLP, sparsity constraints are useful in various applications, such as POS-tagging [30], dependency parsing [16] and document classification [32]. It has been shown that imposing sparse Dirichlet priors in Latent Dirichlet Allocation (LDA) is useful for downstream NLP tasks like POS-tagging [30], and improves interpretability [23]. Experiments show that the gathered descriptions for a given word are typically limited to approximately 20–30 features in norming studies [31].

In this paper, we propose a new, principled sparse representation method that learns sparse overcomplete word representations directly from the raw unlabeled text. We also design an easy-to-parallelize algorithm, which is based on noise-contrastive estimation (NCE) to train our proposed model. Additionally, we propose a clustering-based adaptive updating scheme for noise distributions used by NCE for effective learning. This updating scheme makes the noise distributions approximate the data distribution, and thus pushes the learning improves with fewer noise samples.

We evaluate our model on the word analogy, word similarity, and word intrusion tasks. The first two tasks are used to examine the expressive power of the learned representations and the last task is for interpretability. On the word analogy task, the results show that our proposed sparse model can achieve competitive performance with the state-of-the-art models under the same settings. On the word similarity task, the proposed model outperforms the competitors. For the interpretability, experimental results demonstrate that the sparse word vectors are much more interpretable.

2 Related Work

Learning sparse word vectors is booming along with dense vector representations. We put the existing sparse word vector learning methods into two categories: matrix-based methods and neural network-based methods.

The matrix-based methods can be divided into two steps. The first step is to construct a co-occurrence matrix $\boldsymbol{M}$ of size $V \times C$, where V is the vocabulary size and C is the context size. The w-th row $\boldsymbol{M}_{w,:}$ is the initial representation of the w-th word. The second step is to apply a dimension reduction method to map $\boldsymbol{M}$ to a sparse matrix $\boldsymbol{M}'$ of size $V \times d$, where $d \ll C$. For example, Murphy et al. (2012) improved the interpretability of word vectors by introducing sparse and non-negative constrains into matrix factorization [20]. Levy and Goldberg (2014) showed that the sparse word vectors of word-context co-occurrence PPMI statistics also possess linguistic regularities that present in dense neural embeddings [14].

The neural network-based methods usually transforms dense neural embeddings into sparse ones. These methods generally require two steps of learning procedures. The first step is to train an embedding model, such as CBoW, SkipGram [17] and GloVe [24] to obtain dense feature vectors of the words in the vocabulary. The second step is to learn the sparse representation of each word by fixing the dense word embeddings. For example, Faruqui et al. (2015) transformed dense word vectors into sparse representations using dictionary learning method and showed the resulting sparse vectors are more similar to the interpretable features typically used in NLP [6]. Chen et al. (2016) proposed to use sparse linear combination of common words to represent uncommon ones, which results in sparse representation of words that is effective of compressing neural language models.

In summary, it is tricky to construct the co-occurrence matrix and design the dimension reduction algorithm to obtain good sparse word vectors. For neural network-based methods, the two-step pipelines may lose the sparse structure of words. This is because the prior that each word has a sparse structure is not imposed to learn dense embeddings, which could potentially lose the information for the sparse vector learning. Hence, it is preferred to learn sparse vectors without intermediate dense word embeddings like the method Sun et al. (2016) proposed [29]. The method Sun et al. proposed does not learn overcomplete sparse word vectors.

3 Our Model

In this section, we will talk about a model that try to discover the fundamental elements that constitute each word. We call these fundamental elements *word atoms*. Word atoms and words are analogs to atoms and molecules in Chemistry, respectively. The types of atoms are very small, but they can make up a huge number of different molecules. Likewise, we expect the limited word atoms could represent a large number of words in the vocabulary. A word atom can be regarded as an indivisible semantic or syntactic object.

The design philosophy of our model is similar to that of SkipGram, i.e., "good representations result in good performances to predict context words". In addition, we assume that each word is composed of a few *word atoms*. In detail, each word is assumed to be represented by a sparse, linear combination of word atoms' vectors. This assumption is similar to but different from Chen et al.'s [2], which assume that each uncommon word is represented by a sparse, linear combination of the common ones. A word should not have too many semantic or syntactic components, so the sparseness assumption is reasonable.

Before introducing the mathematical model of representing words, we describe briefly the denotations. Let the vocabulary $\mathcal{V} = \{w_1, w_2, \dots, w_n\}$, contexts $\mathcal{C} = \{c_1, c_2, \dots, c_{n'}\}$. Each column of $\boldsymbol{B} \in \mathbb{R}^{d \times n_b}$, and $\boldsymbol{C} \in \mathbb{R}^{d \times n_c}$ are word and context atoms, respectively.[1] n_b and n_c are the number of word and context atoms, respectively; d is the dimension of atom vectors. For any given word $w \in \mathcal{V}$, its vector representation is $\boldsymbol{w} = \boldsymbol{B\alpha}$, where $\boldsymbol{\alpha}$, called the sparse representation of w, is the coefficients that are used to combine word atoms to make up the word. Similarly, for a context $c \in \mathcal{C}$, its vector representation is $\boldsymbol{c} = \boldsymbol{C\beta}$, where $\boldsymbol{\beta}$ is the sparse representation of c.

We think good word representations are helpful to predict a word's surrounding context words. The *softmax* model is used to model the distribution of a word's surrounding contexts.

$$p(c \mid w) = \frac{\exp(\boldsymbol{w}^\top \boldsymbol{c})}{\sum_{c' \in \mathcal{C}} \exp(\boldsymbol{w}^\top \boldsymbol{c}')} = \frac{\exp(\boldsymbol{\alpha}^\top \boldsymbol{B}^\top \boldsymbol{C\beta})}{\sum_j \exp(\boldsymbol{\alpha}^\top \boldsymbol{B}^\top \boldsymbol{C\beta}_j)}. \tag{1}$$

The word-context pair (w, c) is drawing from plain text. Concretely, for input plain text that consists of N words $w_1, w_2, \dots, w_N$, the word-context pair (w_i, w_j) is drawn such that $|j - i| < \ell/2$, where ℓ is the window size.

It is difficult to train model (1) with the maximum likelihood estimation because of the difficulty of computing the normalization constant (a.k.a. partition function) for each word. In the literature, there are several methods to confront the partition function of a *single distribution*, such as MCMC-based algorithms, pseudo-likelihood, (denoising) score matching, Noise-Contrastive Estimation (NCE) [10]. But not all of these methods can be applied to discrete-input models like (1).

3.1 Parameter Estimation

We will adopt NCE to train model (1). The basic idea of NCE is to train a logistic regression classifier to discriminate between samples from the data distribution and samples from some "noise" distributions. It is a parameter estimation technique that is asymptotically unbiased and is suitable to estimate the parameters of a model with few number of random variables [8]. And it is also applicable to discrete-input models. One issue to apply NCE to train model (1) is that our model is a series of distributions that share the same parameters, which does not accommodate to NCE's setting. Following the work using NCE to train neural language models [2,19] and word embeddings [18], we define the training objective as the expectation of all distributions' NCE objective functions.

[1] The same as a word is composed of word atoms, we also assume that a context is composed of context atoms. In this paper, we will use surrounding words as contexts and thus $\mathcal{V} = \mathcal{C}$. The number of word and context atoms are also set to be equal, i.e., $n_b = n_c$.

In our situation, where the number of estimated conditional distributions is fairly small, we could learn a parameter corresponds to the partition function of each conditional distribution following the standard procedures NCE suggested to handle unnormalized probabilities [10]. Denote these parameters as a vector $\boldsymbol{z} = (z_1, z_2, \ldots, z_V)$. Suppose to draw k negative instances per positive instance. Taking the sparseness requirement on the parameter $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ into consideration, the resulting parameter estimation model for (1) is

$$\arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) - \lambda h(\boldsymbol{S}_\alpha, \boldsymbol{S}_\beta), \tag{2}$$

where $\boldsymbol{\theta} = \{\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{S}_\alpha, \boldsymbol{S}_\beta, \boldsymbol{z}\}$, $\boldsymbol{S}_\alpha = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_V)$, $\boldsymbol{S}_\beta = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_V)$ and

$$J(\boldsymbol{\theta}) = \sum_{(w_i, c_i, s_i) \in \mathcal{D}} \ln \left(\frac{1 - s_i}{2} + s_i \sigma_k(\boldsymbol{\alpha}_i^\top \boldsymbol{B}^\top \boldsymbol{C} \boldsymbol{\beta}_i + z_{w_i} - \ln p_n(w_i)) \right),$$

$$h(\boldsymbol{S}_\alpha, \boldsymbol{S}_\beta) = \sum_{j=1}^{|\mathcal{V}|} (\|\boldsymbol{\alpha}_j\|_1 + \|\boldsymbol{\beta}_j\|_1),$$

where $\sigma_k(x) = 1/(1 + k \cdot \exp(-x))$ is the logistic function parameterized by k; z_{w_i} is a parameter corresponds to the partition function of distribution $p(\cdot \mid w_i)$; $s_i = +1$ and -1 is a variable indicates whether the corresponding instance is extracted from the corpus (namely, positive instances) or drawn from a noise distribution (namely, negative instances); $\mathcal{D}$ is the training dataset, including positive and negative instances; λ is a hyperparameter used to control the degree of sparseness of $\boldsymbol{S}_\alpha$ and $\boldsymbol{S}_\beta$.

The first term of the optimization problem (2), i.e., $J(\boldsymbol{\theta})$, is derived from applying NCE to model (1). The second term—which is a ℓ_1 regularization—encourages sparse solutions for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

Because our goal is not to obtain an accurate prediction model but rather the vector representations of the word atoms and the sparse codes, following Mikolov et al.'s suggestion in [17], we adopt a simplified version of NCE, which called "negative sampling (NS)" to learn word representations. This is done by redefine the first term of model (2) as

$$J(\boldsymbol{\theta}) = \sum_{(w_i, c_i, s_i) \in \mathcal{D}} \log \sigma(s_i \boldsymbol{\alpha}_i^\top \boldsymbol{B}^\top \boldsymbol{C} \boldsymbol{\beta}_i),$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the *sigmoid* function.

Denote model (2) as SpVec ignoring the concrete definition of $J(\boldsymbol{\theta})$. We use a suffix to indicate which training algorithm is applied: when NCE (NS) is used, we call the model SpVec-nce (SpVec-ns). Note that the SpVec model has two sets of word representations: one for target words and one for context words, which is the same as SkipGram. For SkipGram, the word analogy test experiments show that target word embeddings and context word embeddings have similar structures: both embeddings encode the relationship between words by the difference of corresponding words. Additionally, the sparse representation of words is a description of the structure of a word: it determines how a word is composed from word atoms. Therefore, a natural question is that: is it possible to enforce the words and contexts to have the same sparse representations? This can be done by setting $\boldsymbol{S}_\alpha$ and $\boldsymbol{S}_\beta$ to share an identical parameter set, which introduces another variant of SpVec. We denote it with a prefix: s-SpVec, which means the sparse vectors are **s**hared.

Both NS and NCE require some noise distributions to draw negative instances. When NS is used, an identical distribution $p_n(w) \propto \#(w)^{0.75}$ is used. When NCE is used, dynamic distributions are used to draw negative instances, which is inspired by *self-contrastive estimation* (SCE) [9]. According to the theory of NCE, this estimation method can learn effectively when the negative samples are drawing from a distribution similar to the data distribution. The model is approaching the data distribution along with the training. The SCE therefore suggest to copy the trained model as new noise distributions during training. But it is intractable in our scenario, where there are V multinomial distributions, each of which has V parameters to describe. To make it tractable, we apply a clustering method to group the distributions and compute a delegate noise distribution for each group. Concretely, we use M distributions that are updated after every 5% of progress using the following three steps.

1. Compute the dense words and contexts embeddings: $\boldsymbol{U} = \boldsymbol{B}\boldsymbol{S}_\alpha,\ \boldsymbol{V} = \boldsymbol{C}\boldsymbol{S}_\beta$.
2. Apply k-means to cluster word embeddings into M classes.
3. Specify a distribution for every cluster $\mathcal{X}$ using the following formula.

$$P_{\mathcal{X}}(c = i) = \left[\sum_{w \in \mathcal{X}} \frac{\hat{p}_d(w)}{\sum_{w' \in \mathcal{X}} \hat{p}_d(w')} \operatorname{softmax}(\boldsymbol{V}^\top \boldsymbol{w})\right]_i,$$

   where $\hat{p}_d$ is the word frequency distribution; $\operatorname{softmax}(\boldsymbol{y}) = \exp(\boldsymbol{y}) / \sum_i \exp(y_i)$.

The cluster dependent noise distributions have the property that for any $w \in \mathcal{X}$, $p_{\mathcal{X}}(c) \approx p(c \mid w)$. In principle, other clustering methods could also be applied instead of k-means. The k-means clustering method is satisfactory in our experiments.

3.2 Optimization Algorithm

In this subsection, we introduce an easy-to-parallelize algorithm to train SpVec. This algorithm is based on Stochastic Proximal Gradient Descent (SPGD) [22]. Take SpVec-ns as an example.[2] Define the per-instance loss function as $f = -\log \sigma(s\boldsymbol{\alpha}^\top \boldsymbol{B}^\top \boldsymbol{C}\boldsymbol{\beta})$. Suppose the gradients of per-instance loss w.r.t all parameters are obtained, the parameters are updated by the following formulas.

[2] SpVec-nce's learning algorithm could be derived similarly.

$$\boldsymbol{\alpha}^{(t+1)} = \mathrm{prox}_{\eta_t h}\big(\alpha^{(t)} - \eta_t \frac{\partial f}{\partial \boldsymbol{\alpha}}\big), \qquad \boldsymbol{\beta}^{(t+1)} = \mathrm{prox}_{\eta_t h}\big(\beta^{(t)} - \eta_t \frac{\partial f}{\partial \boldsymbol{\beta}}\big), \tag{3}$$

$$\boldsymbol{B}^{(t+1)} = \boldsymbol{B}^{(t)} - \eta_t \frac{\partial f}{\partial \boldsymbol{B}}, \qquad \boldsymbol{C}^{(t+1)} = \boldsymbol{C}^{(t)} - \eta_t \frac{\partial f}{\partial \boldsymbol{C}}, \tag{4}$$

where $h(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$; η_t is the learning rate at the t-th step; $\mathrm{prox}_g(\boldsymbol{x}) = \arg\min_{\boldsymbol{u}} \big(g(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{x}\|_2^2\big)$ is the proximal mapping. In detail,

$$\big(\mathrm{prox}_{\eta h}(\boldsymbol{x})\big)_i = \begin{cases} x_i - \lambda\eta, & x_i > \lambda\eta, \\ 0, & -\lambda\eta \le x_i \le \lambda\eta, \\ x_i + \lambda\eta, & x_i < -\lambda\eta. \end{cases}$$

However, it is inefficient to update the model for every training instance. Therefore, we design an algorithm that updates parameters on mini-batches. The core idea is to update the parameters based on the loss function defined on mini-batches, which is carefully arranged so that the gradients can be expressed by simple matrix-matrix products.

We represent a mini-batch using a vector $\boldsymbol{w} \in \mathbb{N}^m$ and a matrix $\boldsymbol{c} \in \mathbb{N}^{(k+1)\times m}$, where m is the size of mini-batches. The index vector $\boldsymbol{w}$ and the first row of $\boldsymbol{c}$ (denoted by[3] $\boldsymbol{c}_{1,:}$) form a set of positive instances, i.e., a pair $(w_i, c_{1,i})$ is a positive instance. Similarly, $\boldsymbol{w}$ and $\boldsymbol{c}_{i,:}, 1 < i \le k+1$ form negative instances. Denote[4]

$$\begin{aligned} \boldsymbol{s}_{1,:} &= +1 & \boldsymbol{s}_{2:k+1,:} &= -1 \\ \bar{\boldsymbol{\alpha}} &= (\boldsymbol{S}_\alpha)_{:,\boldsymbol{w}} \in \mathbb{R}^{n_b \times m}, & \bar{\boldsymbol{\beta}} &= (\boldsymbol{S}_\beta)_{:,\boldsymbol{c}} \in \mathbb{R}^{n_c \times (k+1) \times m}, \\ \boldsymbol{a} &= \boldsymbol{B}\bar{\boldsymbol{\alpha}} \in \mathbb{R}^{d\times m}, & \boldsymbol{b} &= \boldsymbol{C}\bar{\boldsymbol{\beta}} \in \mathbb{R}^{d\times(k+1)\times m}, \\ \boldsymbol{z}_{:,j} &= \boldsymbol{b}_{:,:,j}^\top \boldsymbol{a}_{:,j}, & \Sigma_{ij} &= \sigma(s_{ij} z_{ij}), \\ \gamma_{ij} &= -s_{ij}(1 - \Sigma_{ij}), & \boldsymbol{Q} &= (\bar{\boldsymbol{\beta}}_{:,:,j}\boldsymbol{\gamma}_{:,j})_{j=1,2,\ldots,m}, \\ \boldsymbol{M} &= \boldsymbol{C}^\top \boldsymbol{a}, & \boldsymbol{N} &= (\boldsymbol{b}_{:,:,j}\boldsymbol{\gamma}_{:,j})_{j=1,2,\ldots,m}, \end{aligned}$$

Redefine f as the loss on mini-batches $f(\bar{\boldsymbol{\alpha}}, \bar{\boldsymbol{\beta}}, \boldsymbol{B}, \boldsymbol{C}) = -\sum_{i=1}^{k+1}\sum_{j=1}^{m} \log \Sigma_{ij}$. We can prove

$$\frac{\partial f}{\partial \boldsymbol{B}} = \boldsymbol{N}\bar{\boldsymbol{\alpha}}^\top, \qquad \frac{\partial f}{\partial \boldsymbol{C}} = \boldsymbol{a}\boldsymbol{Q}^\top, \tag{5}$$

$$\frac{\partial f}{\partial \boldsymbol{a}} = \boldsymbol{N}, \qquad \frac{\partial f}{\partial \bar{\boldsymbol{\alpha}}} = \frac{\partial \boldsymbol{a}}{\partial \bar{\boldsymbol{\alpha}}}\frac{\partial f}{\partial \boldsymbol{a}} = \boldsymbol{B}^\top \boldsymbol{N}, \tag{6}$$

$$\frac{\partial f}{\partial \bar{\boldsymbol{\beta}}_{:,:,j}} = \boldsymbol{M}_{:,j}\boldsymbol{\gamma}_{:,j}^\top. \tag{7}$$

In summery, the mini-batch SPGD is repeatedly applying Eqs. (5)–(7) to calculate the gradients of parameters and using (3)–(4) to update the parameters.

[3] We use index convention from Python except that indexes start with 1.

[4] We define the product of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and a 3-way tensor $\boldsymbol{B} \in \mathbb{R}^{n\times p\times q}$ to be a 3-way tensor $\boldsymbol{C}$ such that $\boldsymbol{C}_{:,i,j} = \boldsymbol{A}\boldsymbol{B}_{:,i,j}$.

Note that there could be duplicated word or context index in $\boldsymbol{w}$ or $\boldsymbol{c}$ and duplicated gradients w.r.t $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ should be combined before updating when the program is running in parallel. All gradients are calculated by matrix products, which means it is easy to leverage existing high performance algebra libraries to parallelize the computations.

4 Evaluation

In this section, we will evaluate the resulting sparse representations on two similarity-based tasks and investigate the interpretability of our SpVec model.

4.1 Experimental Settings

We use the English Wikipedia dump (July, 2014) as the corpus to train all the models. After some preprocessing such as document extraction, markup removing, sentence splitting, tokenization, lowercasing and text normalization, the plain text corpus contains about 1.6 billion running words.

The hyper parameters for SpVec are given as follows. The number of word or context atoms is set to be 1024 and the dimension of these atom embeddings is set to be 200. The ℓ_1 regularization penalty λ was set empirically such that the overall sparsity of words exceeds 95% for all variants of SpVec. The size of mini-batch is set to be 1024. The learning rate is dynamically updated using formula $\alpha = \alpha_0 - (\alpha_0 - \alpha_{\text{end}})g$, where $g \in [0, 1]$ is the training progress, the initial learning rate α_0 and the minimum learning rate α_{end} are set to be 5×10^{-5} and 1×10^{-6}, respectively. Following Mikolov et al.'s work [17], we use windows with random sizes to draw positive instances and the largest distance between a target word and a context word is 8. During training, we draw 8 negative instances for each positive instance.

After training, we perform an extra operation to further increase the sparsity of the sparse representations. This is done by setting the values that is less than a small fraction of the largest element of the vector (in absolute sense) be zero, i.e., setting α_i be 0 if $|\alpha_i| < \xi \cdot \max\{|\alpha_j| : 1 \leq j \leq n_b\}$. In practice, ξ is set to be 0.05.

For SkipGram, CBoW and SC[5], we train them using the released tools on the same corpus with the same settings as our models if possible for fair comparison. The first two models, which are implemented in the `word2vec` tool[6] are both trained with negative sampling since NCE is not implemented in the tool. The PPMI matrix is built based on the word-context co-occurrence counts with window size as 8.

4.2 Word Analogy

The word analogy task can be used to evaluate models' ability to encode linguistic regularities between words, which is introduced by Mikolov et al. [17]. We

[5] https://github.com/mfaruqui/sparse-coding.
[6] https://github.com/dav/word2vec.

Table 1. Results of word analogy and word intrusion tasks. We report accuracy (%) for word analogy task.

Model	Dim.	Sparsity	DistRatio	Google			MSR			
				Sem.	Syn.	Total	Adj.	Nouns	Verbs	Total
SkipGram	200	0.0%	1.11	73.9	70.8	71.9	66.7	63.3	67.5	66.5
CBoW	200	0.0%	1.08	72.2	69.8	70.7	65.8	61.9	66.1	65.2
Sparse CBoW[a]	300	90.1%	1.39	73.2	67.5	70.1	-	-	-	-
SC (SG)[b]	1024	94.3%	1.27	67.4	60.1	62.7	59.6	57.4	58.9	58.8
SC (CBoW)[c]	1024	93.6%	1.21	68.9	63.4	65.4	60.3	57.8	59.2	59.3
PPMI	60000	90.8%	1.31	74.0	40.3	52.3	38.2	35.5	37.7	37.4
SpVec-ns	1024	96.1%	1.44	70.8	68.1	69.1	63.4	60.0	64.7	63.3
SpVec-nce	1024	95.1%	1.46	69.5	68.5	68.9	64.1	62.3	64.4	63.9
s-SpVec-ns	1024	96.3%	1.47	70.5	68.7	69.3	65.2	63.1	64.8	64.6
s-SpVec-ncs	1024	96.5%	1.51	70.1	69.4	69.6	65.0	63.9	65.3	65.0

[a]This line is adopted from [29].

[b]The input matrix of SC is the 200d vectors of SkipGram in the first row.

[c] The input matrix of SC is the 200d vectors of CBoW in the second row.

use two word analogy test sets, namely, Google and MSR, both containing test case like "*run* is to *running* as *walk* is to *walking*". The Google dataset[7] contains 19,544 analogy questions, which can be categorized into semantic and morpho-syntactic related subsets [17]. The MSR dataset[8] contains 8,000 analogy questions, categorized according to part-of-speech; all of them are morpho-syntactic.

This task is to predict the last word of the analogy questions, pretending it is missing. Following Mikolov et al.'s work [17], for question "a is to b as c is to __", we apply $d = \arg\max_{d \in \mathcal{V} \setminus \{a,b,c\}} \cos(\boldsymbol{c} - \boldsymbol{a} + \boldsymbol{b}, \boldsymbol{d})$ to fill the blank. Table 1 shows the result on word analogy tasks. It shows that word analogy is more challenging for sparse models. None of sparse models outperforms SkipGram or CBoW. Nevertheless, SpVec models can achieve similar performance comparing with SkipGram or CBoW. We also find that all variants of SpVec have similar performance on this task.

4.3 Word Similarity

One important indicator to assess the quality of word representations is the clustering property—similar words should have similar vectors. We use WordSim353 dataset [7] to investigate the similarity aspect of the resulting word vectors. This dataset contains 353 word pairs along with their similarity/relatedness scores. We use the sparse word vectors to retrieve and rank the most similar words. For every word w in WordSim353, we rank its similar words by cosine similarities.

[7] https://github.com/dav/word2vec/blob/master/data/questions-words.txt.

[8] http://research.microsoft.com/en-us/projects/rnn/.

The ground truth of w's similar words is a set $\mathcal{U}(w)$, which is a collection of all the words in WordSim353 that the similarity score with w is higher than 0.6.

The recall-precision curve is depicted by Fig. 1. We expect SpVec's curve to be comparable to SC and higher than SkipGram's, which in turn is expected to be higher than PPMI. This means that the similarities induced by SpVec models are more consistent with human cognitions.

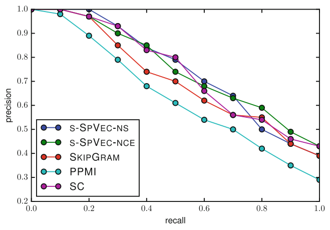

Fig. 1. Recall-precision curve when attempting to rank similar words above unsimilar ones

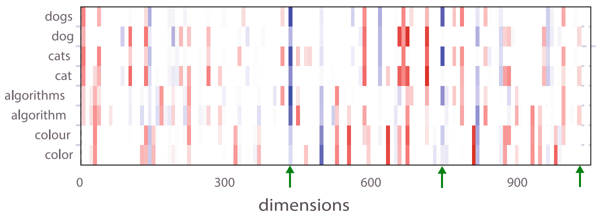

Fig. 2. Visualization of several selected words' sparse representation from s-SpVec-nce. Zeroes are white; negative (positive) values are blue (red). (Color figure online)

4.4 Interpretability

In this subsection, we talk about the interpretability of the learned sparse vectors. We visualize 8 selected words' sparse vectors from s-SpVec-nce in Fig. 2. We find that similar words have similar sparse patterns and dissimilar words possess different sparse codes. We also observe some interpretable patterns from this figure. For example, the dimensions marked by arrows clearly relate to the plural and singular aspects of words.

Following Sun et al.'s work [29], we evaluate the interpretability of our learned sparse word vectors quantitatively by word intrusion task. The details of construction test data for this task are described in [6,29]. Roughly, it sorts words dimensionally and chooses the top 5 and an intruder word to form an instance. An intruder word is a word from the bottom half of the sorted list that is in top 10% of a sorted list corresponds to another dimension.

We use DistRatio to measure the interpretability of word representations. DistRatio is defined to be the average ratio of the distance a_i to distance b_i, where a_i is the average distance between the intruder word and top words for the i-th instance; and b_i is the average distance between the top words for the i-th instance. This measure is first introduced by Sun et al. [29]. The higher the ratio is, the stronger interpretability the representation possesses.

Table 1 presents the DistRatio of our models and their competing ones. It shows that the interpretability of dense models is weak while sparse representations illustrate much stronger interpretability. This confirms that the sparse representations are more interpretable than the dense ones. Moreover, the SpVec variants outperform other sparse representation models significantly on the interpretability aspect. Compared to SC, the reason might be that our method directly learns the sparse word vectors from the data instead of transforming pre-trained dense vectors to sparse codes that SC does, and thus can avoid the information loss caused by the pipeline of learning sparse representations.

5 Conclusion

In this paper, we propose a method to learn sparse word vectors directly from the plain text, which is based on two assumptions: (1) each word is composed of a few fundamental elements and (2) good representations result in good performances to predict context words. We also give an efficient and easy-to-parallelize algorithm that based on NCE to train the proposed model. Additionally, a clustering-based adaptive updating scheme for noise distributions is proposed for effective learning when NCE is applied.

The experimental results on word analogy tasks show that the performance loss due to imposing sparse structure on word representations is limited. On the word similarity task, our models outperform dense representations like Skip-Gram, which is considered to be a strong competitor. On the interpretability aspect, the sparse representations are more interpretable than dense ones. The experiments demonstrate the effectiveness of our learned sparse vectors in interpretability.

Acknowledgement. This research is supported by the National Basic Research Program of China (the 973 Program) under Grant No. 2015CB352201 and the National Natural Science Foundation of China under Grant Nos. 61421091, 61232015 and 61502014.

References

1. Brown, P.F., Della Pietra, V.J., de Souza, P.V., Lai, J.C., Mercer, R.L.: Class-based n-gram models of natural language. Comput. Linguist. **18**(4), 467–479 (1992)
2. Chen, Y., Mou, L., Xu, Y., Li, G., Jin, Z.: Compressing neural language models by sparse word representations. In: Proceedings of ACL (2016)
3. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
4. Donoho, D.L., Elad, M., Temlyakov, V.N.: Stable recovery of sparse overcomplete representations in the presence of noise. IEEE Trans. Inf. Theory **52**, 6–18 (2006)
5. Erk, K., Padó, S.: A structured vector space model for word meaning in context. In: Proceedings of EMNLP, pp. 897–906, Morristown, NJ, USA (2008)
6. Faruqui, M., Tsvetkov, Y., Yogatama, D., Dyer, C., Smith, N.A.: Sparse overcomplete word vector representations. In: Proceedings of ACL, pp. 1491–1500 (2015)

7. Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., Ruppin, E.: Placing search in context: the concept revisited. ACM Trans. Inf. Syst. **20**(1), 116–131 (2002)
8. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
9. Goodfellow, I.J.: On distinguishability criteria for estimating generative models. arXiv, December 2014
10. Gutmann, M., Hyvärinen, A.: Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. J. Mach. Learn. Res. **13**(1), 307–361 (2012)
11. Harris, Z.S.: Distributional structure. Word **10**, 146–162 (1954)
12. Kalchbrenner, N., Blunsom, P.: Recurrent convolutional neural networks for discourse compositionality. arXiv.org, June 2013
13. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Process. **25**(2–3), 259–284 (1998)
14. Levy, O., Goldberg, Y.: Linguistic regularities in sparse and explicit word representations. In: Conference on Natural Language Learning, pp. 171–180 (2014)
15. Lund, K., Burgess, C., Atchley, R.A.: Semantic and associative priming in high-dimensional semantic space. In: Annual Conference of the Cognitive Science Society, vol. 17, pp. 660–665 (1995)
16. Martins, A.F.T., Smith, N.A., Figueiredo, M.A.T., Aguiar, P.M.Q.: Structured sparsity in structured prediction. In: Proceedings of EMNLP (2011)
17. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Computing Research Repository (2013)
18. Mnih, A., Kavukcuoglu, K.: Learning word embeddings efficiently with noise-contrastive estimation. In: NIPS, pp. 2265–2273 (2013)
19. Mnih, A., Teh, Y.W.: A fast and simple algorithm for training neural probabilistic language models. arXiv preprint arXiv:1206.6426 (2012)
20. Murphy, B., Talukdar, P.P., Mitchell, T.M.: Learning effective and interpretable semantic models using non-negative sparse embedding. In: Proceedings of COLING. ACL (2012)
21. Olshausen, B.A., Field, D.J.: Sparse coding with an overcomplete basis set: a strategy employed by v1? Vis. Res. **37**(23), 3311–3325 (1997)
22. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends® Optim. **1**(3), 127–239 (2014)
23. Paul, M., Dredze, M.: Factorial LDA: sparse multi-dimensional text models. In: Advances in Neural Information Processing (2012)
24. Pennington, J., Socher, R., Manning, C.D.: GloVe: global vectors for word representation. In: Proceedings of ACL (2014)
25. Sivaram, G.S.V.S., Nemala, S.K., Elhilali, M., Tran, T.D., Hermansky, H.: Sparse coding for speech recognition. In: ICASSP, pp. 4346–4349 (2010)
26. Socher, R., Bauer, J., Manning, C.D., Ng, A.Y.: Parsing with compositional vector grammars. In: Proceedings of ACL (2013)
27. Socher, R., Chen, D., Manning, C.D.: Reasoning with neural tensor networks for knowledge base completion. In: NIPS (2013)
28. Soutner, D., Müller, L.: Continuous distributed representations of words as input of LSTM network language model. In: Sojka, P., Horák, A., Kopeček, I., Pala, K. (eds.) TSD 2014. LNCS, vol. 8655, pp. 150–157. Springer, Cham (2014). doi:10.1007/978-3-319-10816-2_19

29. Sun, F., Guo, J., Lan, Y., Xu, J., Cheng, X.: Sparse word embeddings using ℓ_1 regularized online learning. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, New York, USA, pp. 959–966 (2016)
30. Toutanova, K., Johnson, M.: A Bayesian LDA-based model for semi-supervised part-of-speech tagging. In: NIPS (2007)
31. Vinson, D.P., Vigliocco, G.: Semantic feature production norms for a large set of objects and events. Behav. Res. Methods **40**(1), 183–190 (2008)
32. Yogatama, D., Smith, N.A.: Linguistic structured sparsity in text categorization. In: Proceedings of ACL (2014)