Neurocomputing ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

PSDVec: A toolbox for incremental and scalable word embedding

Shaohua Li [a,*], Jun Zhu [b], Chunyan Miao [a]

[a] Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), Nanyang Technological University, Singapore
[b] Tsinghua University, PR China

ARTICLE INFO

Article history:
Received 28 July 2015
Received in revised form
28 January 2016
Accepted 25 May 2016
Communicated by Wang Gang

Keywords:
Word embedding
Matrix factorization
Incremental learning

ABSTRACT

PSDVec is a Python/Perl toolbox that learns word embeddings, i.e. the mapping of words in a natural language to continuous vectors which encode the semantic/syntactic regularities between the words. PSDVec implements a word embedding learning method based on a weighted low-rank positive semidefinite approximation. To scale up the learning process, we implement a blockwise online learning algorithm to learn the embeddings incrementally. This strategy greatly reduces the learning time of word embeddings on a large vocabulary, and can learn the embeddings of new words without re-learning the whole vocabulary. On 9 word similarity/analogy benchmark sets and 2 Natural Language Processing (NLP) tasks, PSDVec produces embeddings that has the best average performance among popular word embedding tools. PSDVec provides a new option for NLP practitioners.

1. Introduction

Word embedding has gained popularity as an important unsupervised Natural Language Processing (NLP) technique in recent years. The task of word embedding is to derive a set of vectors in a Euclidean space corresponding to words which best fit certain statistics derived from a corpus. These vectors, commonly referred to as the *embeddings*, capture the semantic/syntactic regularities between the words. Word embeddings can supersede the traditional one-hot encoding of words as the input of an NLP learning system, and can often significantly improve the performance of the system.

There are two lines of word embedding methods. The first line is neural word embedding models, which use softmax regression to fit bigram probabilities and are optimized with Stochastic Gradient Descent (SGD). One of the best known tools is word2vec[1] [10]. The second line is low-rank matrix factorization (MF)-based methods, which aim to reconstruct certain bigram statistics matrix extracted from a corpus, by the product of two low rank factor matrices. Representative methods/toolboxes include Hyperwords[2] [4,5], GloVe[3] [11], Singular[4] [14], and Sparse[5] [2]. All these methods use two different sets of embeddings for words and their context words, respectively. SVD based optimization procedures are used to yield two singular matrices. Only the left singular matrix is used as the embeddings of words. However, SVD operates on $\boldsymbol{G}^{\mathrm{T}}\boldsymbol{G}$, which incurs information loss in $\boldsymbol{G}$, and may not correctly capture the *signed correlations* between words. An empirical comparison of popular methods is presented in [5].

The toolbox presented in this paper is an implementation of our previous work [8]. It is a new MF-based method, but is based on eigendecomposition instead. This toolbox is based on [8], where we establish a Bayesian generative model of word embedding, derive a weighted low-rank positive semidefinite approximation problem to the Pointwise Mutual Information (PMI) matrix, and finally solve it using eigendecomposition. Eigendecomposition avoids the information loss in based methods, and the yielded embeddings are of higher quality than SVD-based methods. However eigendecomposition is known to be difficult to scale up. To make our method scalable to large vocabularies, we exploit the sparsity pattern of the weight matrix and implement a divide-and-conquer approximate solver to find the embeddings incrementally.

Our toolbox is named *Positive-Semidefinite Vectors* (*PSDVec*). It offers the following advantages over other word embedding tools:

1. The incremental solver in PSDVec has a time complexity $O(cd^2n)$ and space complexity $O(cd)$, where n is the number of words in a vocabulary, d is the specified dimensionality of embeddings, and $c \ll n$ is the number of specified core words. Note that the space complexity does not increase with the

* Corresponding author.
E-mail addresses: shaohua@gmail.com (S. Li), dcszj@tsinghua.edu.cn (J. Zhu), ascymiao@ntu.edu.sg (C. Miao).

[1] https://code.google.com/p/word2vec/
[2] https://bitbucket.org/omerlevy/hyperwords
[3] http://nlp.stanford.edu/projects/glove/
[4] https://github.com/karlstratos/singular
[5] https://github.com/mfaruqui/sparse-coding

http://dx.doi.org/10.1016/j.neucom.2016.05.093
0925-2312/

vocabulary. In contrast, other MF-based solvers, including the core embedding generation are of $O(n^3)$ time complexity and $O(n^2)$ space complexity.[6] Hence asymptotically, PSDVec takes about $O(cd^2/n^2)$ of the time and $O(cd/n^2)$ of the space of other MF-based solvers.

2. Given the embeddings of an original vocabulary, PSDVec is able to learn the embeddings of new words incrementally. To our best knowledge, none of other word embedding tools provide this functionality; instead, new words have to be learned together with old words in batch mode. A common situation is that we have a huge general corpus such as English Wikipedia, and also have a small domain-specific corpus, such as the NIPS dataset. In the general corpus, specific terms may appear rarely. It would be desirable to train the embeddings of a general vocabulary on the general corpus, and then incrementally learn words that are unique in the domain-specific corpus. Then this feature of incremental learning could come into play.
3. On word similarity/analogy benchmark sets and common Natural Language Processing (NLP) tasks, PSDVec produces embeddings that has the best average performance among popular word embedding tools.
4. PSDVec is established as a Bayesian generative model [8]. The probabilistic modeling endows PSDVec clear probabilistic interpretation, and the modular structure of the generative model is easy to customize and extend in a principled manner. For example, global factors like topics can be naturally incorporated, resulting in a hybrid model [9] of word embedding and Latent Dirichlet Allocation [1]. For such extensions, PSDVec would serve as a good prototype. While in other methods, the regression objectives are usually heuristic, and other factors are difficult to be incorporated.

2. Problem and solution

PSDVec implements a low-rank MF-based word embedding method. This method aims to fit the $\mathrm{PMI}(s_i, s_j) = \log \frac{P(s_i,s_j)}{P(s_i)P(s_j)}$ using $\boldsymbol{v}_{s_j}^\top \boldsymbol{v}_{s_i}$, where $P(s_i)$ and $P(s_i, s_j)$ are the empirical unigram and bigram probabilities, respectively, and $\boldsymbol{v}_{s_i}$ is the embedding of s_i. The regression residuals $\mathrm{PMI}(s_i, s_j) - \boldsymbol{v}_{s_j}^\top \boldsymbol{v}_{s_i}$ are penalized by a monotonic transformation $f(\cdot)$ of $P(s_i, s_j)$, which implies that, for more frequent (therefore more important) bigram s_i, s_j, we expect it is better fitted. The optimization objective in the matrix form is

$$\boldsymbol{V}^* = \arg\min_{\boldsymbol{V}} \|\boldsymbol{G} - \boldsymbol{V}^\top \boldsymbol{V}\|_{f(\boldsymbol{H})} + \sum_{i=1}^{W} \mu_i \|\boldsymbol{v}_{s_i}\|_2^2, \tag{1}$$

where $\boldsymbol{G}$ is the PMI matrix, $\boldsymbol{V}$ is the embedding matrix, $\boldsymbol{H}$ is the bigram probabilities matrix, $\|\cdot\|_{f(\boldsymbol{H})}$ is the $f(\boldsymbol{H})$– weighted Frobenius-norm, and μ_i are the Tikhonov regularization coefficients. The purpose of the Tikhonov regularization is to penalize overlong embeddings. The overlength of embeddings is a sign of overfitting the corpus. Our experiments showed that, with such regularization, the yielded embeddings perform better on all tasks.

Eq. (1) is to find a weighted low-rank positive semidefinite approximation to $\boldsymbol{G}$. Prior to computing $\boldsymbol{G}$, the bigram probabilities $\{P(s_i, s_j)\}$ are smoothed using Jelinek–Mercer Smoothing.

A Block Coordinate Descent (BCD) algorithm [13] is used to approach (1), which requires eigendecomposition of $\boldsymbol{G}$. The eigendecomposition requires $O(n^3)$ time and $O(n^2)$ space, which is difficult to scale up. As a remedy, we implement an approximate solution that learns embeddings incrementally. The incremental learning proceeds as follows:

1. Partition the vocabulary $\boldsymbol{S}$ into K consecutive groups $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_k$. Take $K=3$ as an example. $\boldsymbol{S}_1$ consists of the most frequent words, referred to as the **core words**, and the remaining words are **noncore words**.
2. Accordingly partition $\boldsymbol{G}$ into $K \times K$ blocks as

$$\begin{pmatrix} \boldsymbol{G}_{11} & \boldsymbol{G}_{12} & \boldsymbol{G}_{13} \\ \boldsymbol{G}_{21} & \boldsymbol{G}_{22} & \boldsymbol{G}_{23} \\ \boldsymbol{G}_{31} & \boldsymbol{G}_{32} & \boldsymbol{G}_{33} \end{pmatrix}.$$

   Partition $f(\boldsymbol{H})$ in the same way. $\boldsymbol{G}_{11}, f(\boldsymbol{H})_{11}$ correspond to **core–core bigrams** (consisting of two core words). Partition $\boldsymbol{V}$ into

$$\begin{pmatrix} \underbrace{\boldsymbol{V}_1}_{\boldsymbol{S}_1} & \underbrace{\boldsymbol{V}_2}_{\boldsymbol{S}_2} & \underbrace{\boldsymbol{V}_3}_{\boldsymbol{S}_3} \end{pmatrix}.$$

3. For core words, set $\mu_i = 0$, and solve $\arg\min_{\boldsymbol{V}} \|\boldsymbol{G}_{11} - \boldsymbol{V}_1^\top \boldsymbol{V}_1\|_{f(\boldsymbol{H}_1)}$ using eigendecomposition, obtaining core embeddings $\boldsymbol{V}_1^*$.
4. Set $\boldsymbol{V}_1 = \boldsymbol{V}_1^*$, and find $\boldsymbol{V}_2^*$ that minimizes the total penalty of the 12th and 21th blocks (the 22th block is ignored due to its high sparsity):

$$\arg\min_{\boldsymbol{V}_2} \|\boldsymbol{G}_{12} - \boldsymbol{V}_1^\top \boldsymbol{V}_2\|_{f(\boldsymbol{H})_{12}}^2 + \|\boldsymbol{G}_{21} - \boldsymbol{V}_2^\top \boldsymbol{V}_1\|_{f(\boldsymbol{H})_{21}}^2 + \sum_{s_i \in \boldsymbol{S}_2} \mu_i \|\boldsymbol{v}_{s_i}\|^2.$$

   The columns in $\boldsymbol{V}_2$ are independent, thus for each $\boldsymbol{v}_{s_i}$, it is a separate weighted ridge regression problem, which has a closed-form solution.
5. For any other set of noncore words $\boldsymbol{S}_k$, find $\boldsymbol{V}_k^*$ that minimizes the total penalty of the 1kth and k1th blocks, ignoring all other kjth and jkth blocks.
6. Combine all subsets of embeddings to form $\boldsymbol{V}^*$. Here $\boldsymbol{V}^* = (\boldsymbol{V}_1^*, \boldsymbol{V}_2^*, \boldsymbol{V}_3^*)$.

3. Software architecture and functionalities

Our toolbox consists of 4 Python/Perl scripts: `extractwiki.py`, `gramcount.pl`, `factorize.py` and `evaluate.py`. Fig. 1 presents the overall architecture.

1. `extractwiki.py` first receives a Wikipedia snapshot as input; it then removes non-textual elements, non-English words and

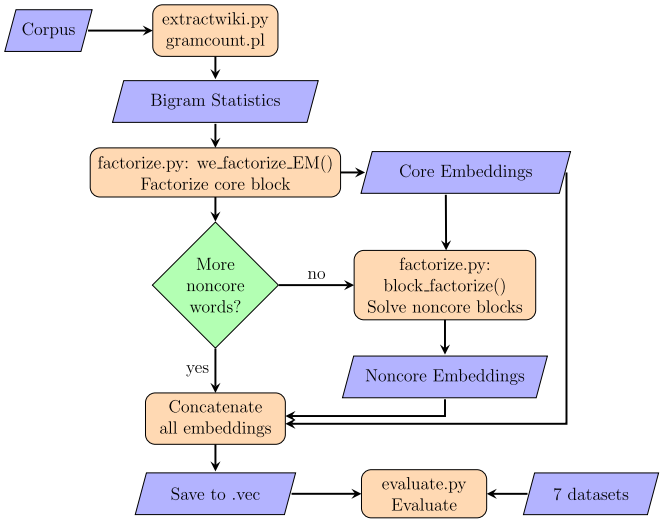

Fig. 1. Toolbox architecture.

[6] Word2vec adopts an efficient SGD sampling algorithm, whose time complexity is only $O(kL)$, and space complexity $O(n)$, where L is the number of word occurrences in the input corpus, and k is the number of negative sampling words, typically in the range 5~20.

Table 1
Performance of each method across 9 tasks.

Method	Similarity tasks							Analogy tasks		NLP tasks		Avg.
	WS	WR	MEN	Turk	SL	TFL	RG	Google	MSR	NER	Chunk	
word2vec	74.1	54.8	73.2	**68.0**	37.4	85.0	81.1	**72.3**	**63.0**	**84.8**	94.8	71.7
PPMI	73.5	67.8	71.7	65.9	30.8	70.0	70.8	52.4	21.7	N.A.[a]	N.A.[a]	58.3
SVD	69.2	60.2	70.7	49.1	28.1	57.5	71.8	24.0	11.3	81.2	94.1	56.1
GloVe	75.9	63.0	75.6	64.1	36.2	87.5	77.0	54.4	43.5	84.5	94.6	68.8
Singular	76.3	**68.4**	74.7	58.1	34.5	78.8	80.7	50.8	39.9	83.8	94.8	67.3
Sparse	74.8	56.5	74.2	67.6	38.4	**88.8**	81.6	71.6	61.9	78.8	**94.9**	71.7
PSDVec	**79.2**	67.9	**76.4**	67.6	**39.8**	87.5	**83.5**	62.3	50.7	84.7	94.7	**72.2**
PSD-unreg[b]	78.6	66.3	75.3	67.5	37.2	85.0	79.9	59.8	46.8	84.7	94.5	70.5

[a] These two experiments are impractical for "PPMI", as they use embeddings as features, and the dimensionality of a PPMI embedding equals the size of the vocabulary, which is over 40,000.
[b] "PSDVec" with all Tikhonov regularization coefficients $\mu_i = 0$, i.e., unregularized.

punctuation; after converting all letters to lowercase, it finally produces a clean stream of English words.
2. `gramcount.pl` counts the frequencies of either unigrams or bigrams in a word stream, and saves them into a file. In the unigram mode (`-m1`), unigrams that appear less than certain frequency threshold are discarded. In the bigram mode (`-m2`), each pair of words in a text window (whose size is specified by `-n`) forms a bigram. Bigrams starting with the same leading word are grouped together in a row, corresponding to a row in matrices $\boldsymbol{H}$ and $\boldsymbol{G}$.
3. `factorize.py` is the core module that learns embeddings from a bigram frequency file generated by `gramcount.pl`. A user chooses to split the vocabulary into a set of core words and a few sets of noncore words. `factorize.py` can: (1) in function `we_factorize_EM()`, do BCD on the PMI submatrix of core-core bigrams, yielding *core embeddings*; (2) given the core embeddings obtained in (1), in `block_factorize()`, do a weighted ridge regression w.r.t. *noncore embeddings* to fit the PMI submatrices of core-noncore bigrams. The Tikhonov regularization coefficient μ_i for a whole noncore block can be specified by `-t`. A good rule-of-thumb for setting μ_i is to increase μ_i as the word frequencies decrease, i.e., give more penalty to rarer words, since the corpus contains insufficient information of them.
4. `evaluate.py` evaluates a given set of embeddings on 7 commonly used testsets, including 5 similarity tasks and 2 analogy tasks.

4. Implementation and empirical results

4.1. Implementation details

The Python scripts use Numpy for the matrix computation. Numpy automatically parallelizes the computation to fully utilize a multi-core machine.

The Perl script `gramcount.pl` implements an embedded C++ engine to speed up the processing with a smaller memory footprint.

4.2. Empirical results

Our competitors include: **word2vec**, **PPMI** and **SVD** in Hyperwords, **GloVe**, **Singular** and **Sparse**. In addition, to show the effect of Tikhonov regularization on "PSDVec", evaluations were done on an unregularized PSDVec (by passing "`-t0`" to `factorize.py`), denoted as **PSD-unreg**. All methods were trained on an 12-core Xeon 3.6 GHz PC with 48 GB of RAM.

We evaluated all methods on two types of testsets. The first type of testsets are shipped with our toolkit, consisting of 7 word similarity tasks and 2 word analogy tasks (Luong's Rare Words is excluded due to many rare words contained). Seven out of the 9 testsets are used in [5]. The hyperparameter settings of other methods and evaluation criteria are detailed in [5,14,2]. The other 2 tasks are TOEFL Synonym Questions (**TFL**) [3] and Rubenstein & Goodenough (**RG**) dataset [12]. For these tasks, all 7 methods were trained on the April 2015 English Wikipedia. All embeddings except "Sparse" were 500 dimensional. "Sparse" needs more dimensionality to cater for vector sparsity, so its dimensionality was set to 2500. It used the embeddings of word2vec as the input. In analogy tasks `Google` and `MSR`, embeddings were evaluated using 3CosMul [6]. The embedding set of PSDVec for these tasks contained 180,000 words, which was trained using the blockwise online learning procedure described in Section 5, based on 25,000 core words.

The second type of testsets are 2 practical NLP tasks for evaluating word embedding methods as used in [15], i.e., Named Entity Recognition (**NER**) and Noun Phrase Chunking (**Chunk**). Following settings in [15], the embeddings for NLP tasks were trained on Reuters Corpus, Volume 1 [7], and the embedding dimensionality was set to 50 ("Sparse" had a dimensionality of 500). The embedding set of PSDVec for these tasks contained 46,409 words, based on 15,000 core words.

Table 1 above reports the performance of 7 methods on 11 tasks. The last column reports the average score. "PSDVec" performed stably across the tasks, and achieved the best average score. On the two analogy tasks, "word2vec" performed far better than all other methods (except "Sparse", as it was derived from "word2vec"), the reason for which is still unclear. On NLP tasks, most methods achieved close performance. "PSDVec" outperformed "PSD-unreg" on all tasks.

To compare the efficiency of each method, we presented the training time of different methods across 2 training corpora in Table 2. Note that the ratio of running time is determined by a few factors together: the ratio of vocabulary sizes (180,000/

Table 2
Training time (minutes) of each method across 2 training corpora.

Method	Language	Wikipedia	RCV1	Ratio
word2vec	C	249	15	17
PPMI	Python	2196	57	39
SVD	Python	2282	58	39
GloVe	C	229	6	38
Singular	C++	**183**	26	7
Sparse	C++	1548	**1**	1548
PSDVec	Python	463	34	14
PSD-core[a]	*Python*	*137*	*31*	*4*

[a] This is the time of generating the core embeddings only, and is not comparable to other methods.

Please cite this article as: S. Li, et al., `PSDVec`: A toolbox for incremental and scalable word embedding, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2016.05.093

Table 3
Efficiency of incremental learning of PSDVec.

Method	Wikipedia ($c = 25,000$, $d = 500$)					RCV1 ($c = 15,000$, $d = 50$)				
	Words	Time (min)	RAM (G)	Words/min	Speedup	Words	Time (min)	RAM (G)	Words/min	Speedup
PSD-noncore	155,000	326	22	475	2.6	31,409	3	8	10,000	20
PSD-core	25,000	137	44	182	/	15,000	31	15	500	/

46,409 ≈ 4), the ratio of vector lengths (500/50 = 10), the language efficiency, and the algorithm efficiency. We were most interested in the algorithm efficiency. To reduce the effect of different language efficiency of different methods, we took the ratio of the two training time to measure the scalability of each algorithm.

From Table 2, we can see that "PSDVec" exhibited a competitive absolute speed, considering the inefficiency of Python relative to C/C++. The scalability of "PSDVec" ranked the second best, worse than "Singular" and better than "word2vec".

The reason that "PPMI" and "SVD" (based on "PPMI") were so slow is that "hyperwords" employs an external sorting command, which is extremely slow on large files. The reason for the poor scalability of "Sparse" is unknown.

Table 3 shows the time and space efficiency of the incremental learning ("PSD-noncore" for noncore words) and MF-based learning ("PSD-core" for core words) on two corpora. The memory is halved using incremental learning, and is constant as the vocabulary size increases. As stated above, the per-word time complexity of "PSD-noncore" is $O(c^{2.4}d)$. The embedding dimensionality on Wikipedia is 10 times of that on RCV1, thus the speedup on the Wikipedia corpus is only around 1/8 of that on the RCV1 corpus.

5. Illustrative example: training on English Wikipedia

In this example, we train embeddings on the English Wikipedia snapshot in April 2015. The training procedure goes as follows:

1. Use extractwiki.py to cleanse a Wikipedia snapshot, and generate cleanwiki.txt, which is a stream of 2.1 billion words.
2. Use gramcount.pl with cleanwiki.txt as input, to generate top1grams-wiki.txt.
3. Use gramcount.pl with top1grams-wiki.txt and cleanwiki.txt as input, to generate top2grams-wiki.txt.
4. Use factorize.py with top2grams-wiki.txt as input, to obtain 25,000 core embeddings, saved into 25000-500-EM.vec.
5. Use factorize.py with top2grams-wiki.txt and 25000-500-EM.vec as input, and Tikhonov regularization coefficient set to 2, to obtain 55,000 noncore embeddings. The word vectors of totally 80,000 words is saved into 25000-80000-500-BLKEM.vec.
6. Repeat Step 5 twice with Tikhonov regularization coefficient set to 4 and 8, respectively, to obtain extra 50,000 × 2 noncore embeddings. The word vectors are saved into 25000-130000-500-BLKEM.vec and 25000-180000-500-BLKEM.vec, respectively.
7. Use evaluate.py to test 25000-180000-500-BLKEM.vec.

6. Conclusions

We have developed a Python/Perl toolkit `PSDVec` for learning word embeddings from a corpus. This open-source cross-platform software is easy to use, easy to extend, scales up to large vocabularies, and can learn new words incrementally without retraining the whole vocabulary. The produced embeddings performed stably on various test tasks, and achieved the best average score among 7 state-of-the-art methods.

Acknowledgments

This research is supported by the National Research Foundation Singapore under its Interactive Digital Media (IDM) Strategic Research Programme. Part of the work was conceived when Shaohua Li was visiting Tsinghua. Jun Zhu is supported by National NSF of China (No. 61322308) and the Youth Top-notch Talent Support Program.

References

[1] David M. Blei, Andrew Y. Ng, Michael I. Jordan, Latent Dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.
[2] Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, Noah A. Smith, Sparse overcomplete word vector representations, in: Proceedings of ACL, July 26-31, 2015, Beijing, China, Vol. 1: Long Papers, p. 1491–1500.
[3] Thomas K. Landauer, Susan T. Dumais, A solution to Plato's problem: the latent semantic analysis theory of acquisition, induction, and representation of knowledge, Psychol. Rev. 104 (2) (1997) 211.
[4] Omer Levy, Yoav Goldberg, Neural word embeddings as implicit matrix factorization, in: Proceedings of NIPS 2014, December 8-13, 2014, Montreal, Quebec, Canada.
[5] Omer Levy, Yoav Goldberg, Ido Dagan, Improving distributional similarity with lessons learned from word embeddings, Trans. Assoc. Comput. Linguist. 3 (2015) 211–225.
[6] Omer Levy, Yoav Goldberg, Israel Ramat-Gan, Linguistic regularities in sparse and explicit word representations, in: Proceedings of CoNLL-2014, June 26-27, Baltimore, Maryland, USA, 2014, p. 171.
[7] David D. Lewis, Yiming Yang, Tony G. Rose, Fan Li, Rcv1: a new benchmark collection for text categorization research, J. Mach. Learn. Res. 5 (2004) 361–397.
[8] Shaohua Li, Jun Zhu, Chunyan Miao, A generative word embedding model and its low rank positive semidefinite solution, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, September 2015, Association for Computational Linguistics, pp. 1599–1609.
[9] Shaohua Li, Jun Zhu, Chunyan Miao, Generative Topic Embedding: A continuous Representation of Documents, To appear in the Proceedings of The 54th Annual Meeting of the Association for Computational Linguistics (ACL), August 7-12, 2016, Berlin, Germany.
[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, Jeff Dean, Distributed representations of words and phrases and their compositionality, in: Proceedings of NIPS 2013, December 5-8, 2013, Lake Tahoe, Nevada, United States, pp. 3111–3119.
[11] Jeffrey Pennington, Richard Socher, Christopher D. Manning, Glove: global vectors for word representation, in: Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014), October 25-29, 2014, Doha, Qatar, vol. 12, 2014.
[12] Herbert Rubenstein, John B. Goodenough, Contextual correlates of synonymy, Commun. ACM 8 (October (10)) (1965) 627–633.
[13] Nathan Srebro, Tommi Jaakkola, Weighted low-rank approximations, in: Proceedings of ICML 2003, August 21-24, 2003, Washington, DC, USA, vol. 3, 2003, pp. 720–727.
[14] Karl Stratos, Michael Collins, Daniel Hsu, Model-based word embeddings from decompositions of count matrices, in: Proceedings of ACL 2015, July 26-31, 2015, Beijing, China.
[15] Joseph Turian, Lev Ratinov, Yoshua Bengio, Word representations: a simple and general method for semi-supervised learning, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden. pp. 384–394.

Shaohua Li is currently a Ph.D. student in the School of Computer Engineering, Nanyang Technological University (NTU), Singapore. He obtained his B.Sc. in Applied Mathematics from USTC in 2002, and his Master's degree in Computer Engineering from Institute of Software, CAS in 2005. His research interests include Bayesian inference, unsupervised learning and distributed representations of natural languages.

Jun Zhu is an associate professor of Computer Science at Tsinghua University, and an adjunct associate professor of Machine Learning at Carnegie Mellon University. His research focuses on developing statistical machine learning methods to understand complex scientific and engineering data. His current interests are in latent variable models, large-margin learning, Bayesian nonparametrics, and sparse learning in high dimensions. Before joining Tsinghua in 2011, he was a post-doc researcher and project scientist at the Machine Learning Department in Carnegie Mellon University.

Chunyan Miao is an associate professor in the School of Computer Engineering (SCE) at Nanyang Technological University (NTU). She is the director of the NTU-UBC Joint Research Centre of Excellence in Active Living for the Elderly (LILY). Prior to joining NTU, she was an instructor and postdoctoral fellow at the School of Computing, Simon Fraser University, Canada. Her major research focus is on studying the cognitive and social characteristics of intelligent agents in multi-agent and distributed AI/CI systems, such as trust, emotions, motivated learning, ecological and organizational behavior. She has made significant contributions in the integration of the above research into emerging technologies such as interactive digital media (e.g., virtual world, social networks, and massively multi-player game), cloud computing, mobile communication, and humanoid robots.

Please cite this article as: S. Li, et al., PSDVec: A toolbox for incremental and scalable word embedding, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2016.05.093