Real-valued Syntactic Word Vectors (RSV) for Greedy Neural Dependency Parsing

Ali Basirat and Joakim Nivre
Department of Linguistics and Philology
Uppsala University
{ali.basirat,joakim.nivre}@lingfil.uu.se

Abstract

We show that a set of real-valued word vectors formed by right singular vectors of a transformed co-occurrence matrix are meaningful for determining different types of dependency relations between words. Our experimental results on the task of dependency parsing confirm the superiority of the word vectors to the other sets of word vectors generated by popular methods of word embedding. We also study the effect of using these vectors on the accuracy of dependency parsing in different languages versus using more complex parsing architectures.

1 Introduction

Greedy transition-based dependency parsing is appealing thanks to its efficiency, deriving a parse tree for a sentence in linear time using a discriminative classifier. Among different methods of classification used in a greedy dependency parser, neural network models capable of using real-valued vector representations of words, called *word vectors*, have shown significant improvements in both accuracy and speed of parsing. It was first proposed by Chen and Manning (2014) to use word vectors in a 3-layered feed-forward neural network as the core classifier in a transition-based dependency parser. The classifier is trained by the standard back-propagation algorithm. Using a limited number of features defined over a certain number of elements in a parser configuration, they could build an efficient and accurate parser, called the Stanford neural dependency parser. This architecture then was extended by Straka et al. (2015) and Straka et al. (2016). `Parsito` (Straka et al., 2015) adds a search-based oracle and a set of morphological features to the original architecture in order to make it capable of parsing the corpus of universal dependencies. `UDPipe` (Straka et al., 2016) adds a beam search decoding to `Parsito` in order to improve the parsing accuracy at the cost of decreasing the parsing speed.

We propose to improve the parsing accuracy in the architecture introduced by Chen and Manning (2014) through using more informative word vectors. The idea is based on the greedy nature of the back-propagation algorithm which makes it highly sensitive to the initial state of the algorithm. Thus, it is expected that more qualified word vectors positively affect the parsing accuracy. The word vectors in our approach are formed by right singular vectors of a matrix returned by a transformation function that takes a probability co-occurrence matrix as input and expand the data massed around zero. We show how the proposed method is related to `HPCA` (Lebret and Collobert, 2014) and `GloVe` (Pennington et al., 2014).

Using these word vectors with the Stanford parser we could obtain the parsing accuracy of 93.0% UAS and 91.7% LAS on Wall Street Journal (Marcus et al., 1993). The word vectors consistently improve the parsing models trained with different types of dependencies in different languages. Our experimental results show that parsing models trained with Stanford parser can be as accurate or in some cases more accurate than other parsers such as `Parsito`, and `UDPipe`.

2 Transition-Based Dependency Parsing

A greedy transition-based dependency parser derives a parse tree from a sentence by predicting a sequence of transitions between a set of configurations characterized by a triple $c = (\Sigma, B, A)$, where Σ is a stack that stores partially processed nodes, B is a buffer that stores unprocessed nodes in the input sentence, and A is a set of partial parse trees assigned to the processed nodes. Nodes are positive integers corresponding to linear positions of words in the input sentence. The process

Proceedings of the 21st Nordic Conference of Computational Linguistics, pages 20–28,
Gothenburg, Sweden, 23-24 May 2017. ©2017 Linköping University Electronic Press

of parsing starts from an initial configuration and ends with some terminal configuration. The transitions between configurations are controlled by a classifier trained on a history-based feature model which combines features of the partially built dependency tree and attributes of input tokens.

The arc-standard algorithm (Nivre, 2004) is among the many different algorithms proposed for moving between configurations. The algorithm starts with the initial configuration in which all words are in B, Σ is empty, and A holds an artificial node 0. It uses three actions *Shift*, *Right-Arc*, and *Left-Arc* to transition between the configurations and build the parse tree. Shift pushes the head node in the buffer into the stack unconditionally. The two actions Left-Arc and Right-Arc are used to build left and right dependencies, respectively, and are restricted by the fact that the final dependency tree has to be rooted at node 0.

3 Stanford Dependency Parser

The Stanford dependency parser can be considered as a turning point in the history of greedy transition-based dependency parsing. The parser could significantly improve both the accuracy and speed of dependency parsing. The key success of the parser can be summarized in two points: 1) accuracy is improved by using a neural network with pre-trained word vectors, and 2) efficiency is improved by pre-computation to keep the most frequent computations in the memory.

The parser is an arc-standard system with a feed-forward neural-network as its classifier. The neural network consists of three layers: An input layer connects the network to a configuration through 3 real-valued vectors representing words, POS tags, and dependency relations. The vectors that represent POS-tags and dependency relations are initialized randomly but those that represent words are initialized by word vectors systematically extracted from a corpus. Each of these vectors are independently connected to the hidden layer of the network through three distinct weight matrices. A cube activation function is used in the hidden layer to model the interactions between the elements of the vectors. The activation function resembles a third degree polynomial kernel that enables the network to take different combinations of vector elements into consideration. The output layer generates probabilities for decisions between different actions in the arc-standard system. The network is trained by the standard back-propagation algorithm that updates both network weights and vectors used in the input layer.

4 Word Vectors

Dense vector representations of words, in this paper known as word vectors, have shown great improvements in natural language processing tasks. An advantage of this representation compared to the traditional one-hot representation is that the word vectors are enriched with information about the distribution of words in different contexts.

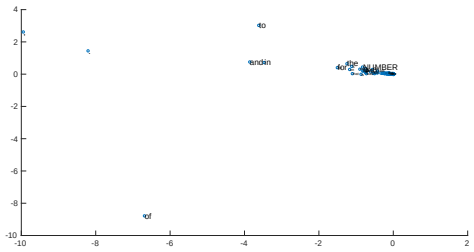
Following Lebret and Collobert (2014), we propose to extract word vectors from a co-occurrence matrix as follows: First we build a co-occurrence matrix $\mathbf{C}$ from a text. The element $\mathbf{C}_{i,j}$ is a maximum likelihood estimation of the probability of seeing word w_j in the context of word w_i, (i.e., $\mathbf{C}_{i,j} = p(w_j|w_i)$. It results in a sparse matrix whose data are massed around zero because of the disproportional contribution of the high frequency words in estimating the co-occurrence probabilities. Each column of $\mathbf{C}$ can be seen as a vector in a high-dimensional space whose dimensions correspond to the context words. In practice, we need to reduce the dimensionality of these vectors. This can be done by standard methods of dimensionality reduction such as principal component analysis. However, the high density of data around zero and the presence of a small number of data points far from the data mass can lead to some meaningless discrepancies between word vectors.

In order to have a better representation of the data we expand the probability values in $\mathbf{C}$ by skewing the data mass from zero toward one. This can be done by any monotonically increasing concave function that magnifies small numbers in its domain while preserving the given order. Commonly used transformation functions with these characteristics are the logarithm function, the hyperbolic tangent, the power transformation, and the Box-cox transformation with some specific parameters. Fig. 1 shows how a transformation function expands high-dimensional word vectors massed around zero.

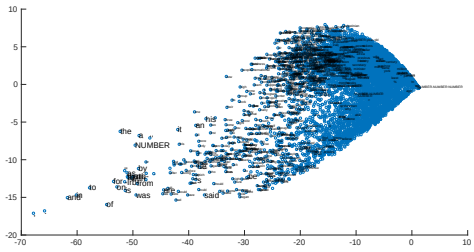
After applying f on $\mathbf{C}$, we centre the column vectors in $f(\mathbf{C})$ around their mean and build the word vectors as below:

$$\Upsilon = \gamma \mathbf{V}_{n,k}^{T} \tag{1}$$

where Υ is a matrix of k dimensional word vectors associated with n words, $\mathbf{V}_{n,k}^{T}$ is the top k



(a)



(b)

Figure 1: PCA visualization of the high-dimensional column-vectors of a co-occurrence matrix (a) before and (b) after applying the transformation function $\sqrt[10]{x}$

right singular vectors of $f(\mathbf{C})$, and $\gamma = \lambda\sqrt{n}$ is a constant factor to scale the unbounded data in the word vectors. In the following, we will refer to our model as RSV, standing for Right Singular word Vector or Real-valued Syntactic word Vectors as it is mentioned in the title of this paper.

5 Experimental Setting

We restrict our experiments to three languages, English, Swedish, and Persian. Our experiments on English are organized as follows: Using different types of transformation functions, we first extract a set of word vectors that gives the best parsing accuracy on our development set. Then we study the effect of dimensionality on parsing performance. Finally, we give a comparison between our best results and the results obtained from other sets of word vectors generated with popular methods of word embedding. Using the best transformation function obtained for English, we extract word vectors for Swedish and Persian. These word vectors are then used to train parsing models on the corpus of universal dependencies.

The English word vectors are extracted from a corpus consisting of raw sentences in Wall Street Journal (WSJ) (Marcus et al., 1993), English Wikicorpus,[1] Thomson Reuters Text Research Collection (TRC2), English Wikipedia corpus,[2], and the Linguistic Data Consortium (LDC) corpus. We concatenate all the corpora and split the sentences by the OpenNLP sentence splitting tool.The Stanford tokenizer is used for tokenization. Word vectors for Persian are extracted from the Hamshahri Corpus (AleAhmad et al., 2009), Tehran Monolingual Corpus,[3]and Farsi Wikipedia download from Wikipedia Monolingual Corpora.[4] The Persian text normalizer tool (Seraji, 2015) is used for sentence splitting and tokenization.[5]Word vectors for Swedish are extracted from Swedish Wikipedia available at Wikipedia Monolingual Corpora, Swedish web news corpora (2001-2013) and Swedish Wikipedia corpus collected by Sprkbanken. [6] The OpenNLP sentence splitter and tokenizer are used for normalizing the corpora.

We replace all numbers with a special token NUMBER and convert uppercase letters to lowercase forms in English and Swedish. Word vectors are extracted only for the unique words appearing at least 100 times. We choose the cut-off word frequency of 100 because it is commonly used as a standard threshold in the other references. The 10 000 most frequent words are used as context words in the co-occurrence matrix. Table 1 represents some statistics of the corpora.

	#Tokens	$\#W \geq 1$	$\#W \geq 100$	#Sents
English	8×10^9	14 462 600	404 427	4×10^8
Persian	4×10^8	1 926 233	60 718	1×10^7
Swedish	6×10^8	5 437 176	174 538	5×10^7

Table 1: Size of the corpora from which word vectors are extracted; #Tokens: total number of tokens; #W≥k: number of unique words appearing at least k times in the corpora; #Sents: number of sentences.

The word vectors are evaluated with respect to the accuracy of parsing models trained with them using the Stanford neural dependency parser (Chen and Manning, 2014). The English parsing models are trained and evaluated on the corpus of universal dependencies (Nivre et al., 2016) version 1.2 (UD) and Wall Street Journal (WSJ) (Marcus et al., 1993) annotated with Stanford typed dependencies (SD) (De Marneffe and Manning, 2010) and CoNLL syntactic dependencies (CD) (Johans-

[1] http://www.cs.upc.edu/~nlp/wikicorpus/

[2] https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2-rss.xml

[3] http://ece.ut.ac.ir/system/files/NLP/Resources/

[4] http://linguatools.org/tools/corpora/wikipedia-monolingual-corpora

[5] http://stp.lingfil.uu.se/~mojgan

[6] https://spraakbanken.gu.se/eng/resources/corpus

son and Nugues, 2007). We split WSJ as follow: sections 02–21 for training, section 22 for development, and section 23 as test set. The Stanford conversion tool (De Marneffe et al., 2006) and the LTH conversion tool[7] are used for converting constituency trees in WSJ to SD and CD. The Swedish and Persian parsing models are trained on the corpus of universal dependencies. All the parsing models are trained with gold POS tags unless we clearly mention that predicted POS tags are used.

6 Results

In the following we study how word vectors generated by RSV influence parsing performance. RSV has four main tuning parameters: 1) the context window, 2) the transformation function f, 3) the parameter λ used in the normalization step, and 4) the number of dimensions. The context window can be symmetric or asymmetric with different length. We choose the asymmetric context window with length 1 i.e., the first preceding word, as it is suggested by Lebret and Collobert (2015) for syntactic tasks. λ is a task dependent parameter that controls the variance of word vectors. In order to find the best value of λ, we train the parser with different sets of word vectors generated randomly by Gaussian distributions with zero mean and isotropic covariance matrices $\lambda\mathbf{I}$ with values of $\lambda \in (0, 1]$. Fig. 2a shows the parsing accuracies obtained from these word vectors. The best results are obtained from word vectors generated by $\lambda = 0.1$ and $\lambda = 0.01$. The variation in the results shows the importance of the variance of word vectors on the accuracy of parser. Regarding this argument, we set the normalization parameter λ in Eq. 1 equal to 0.1. The two remaining parameters are explored in the following subsections.

6.1 Transformation Function

We have tested four sets of transformation functions on the co-occurrence matrix:

- $f_1 = \tanh(nx)$
- $f_2 = \sqrt[n]{x}$
- $f_3 = n(\sqrt[n]{x} - 1)$
- $f_4 = \frac{\log(2^{n+1}x+1)}{\log(2^{n+1}+1)}$

where $x \in [0, 1]$ is an element of the matrix and n is a natural number that controls the degree of skewness, the higher the value of n is, the more the data will be skewed. Fig. 2b shows the effect of using these transformation functions on parsing accuracy. Best results are obtained from n^{th}-root and Box-cox transformation functions with $n = 7$, which are closely related to each other. Denoting the set of word vectors obtained from the transformation function f as $\Upsilon(f)$, it can be shown that $\Upsilon(f_3) = \Upsilon(f_2) - n$, since the effect of coefficient n in the first term of f_3 is cancelled out by the right singular vectors in Eq. 1.

Fig. 2c visualizes best transformation functions in each of the function sets. All the functions share the same property of having relatively high derivatives around 0 which allows of skewing data in the co-occurrence matrix to right (i.e., close to one) and making a clear gap between the syntactic structures that can happen (i.e., non-zeros probabilities in the co-occurrence matrix) and those that cannot happen (i.e., zero probabilities in the co-occurrence matrix. This movement from zero to one, however, can lead to disproportional expansions between the data close to zero and those that are close to one. It is because the limit of the ratio of the derivative of $f_i(x)$ $i = 1, 2, 3, 4$ to the derivative of $f_i(y)$ as x approaches to 0 and y approaches to 1 is infinity i.e., $\lim_{x\to 0, y\to 1} \frac{f_i'(x)}{f_i'(y)} = \infty$. The almost uniform behaviour of $f_1(x)$ for $x > 0.4$ results in a small variance in the generated data that will be ignored by the subsequent singular value decomposition step and consequently loses the information provided with the most probable context words. Our solution to these problems is to use the following piecewise transformation function f:

$$f = \begin{cases} \tanh(7x) & x \leq \theta \\ \sqrt[7]{x} & x > \theta \end{cases} \quad (2)$$

where $\theta = 10^{-n}$ and $n \in \mathbb{N}$. This function expands the data in a more controlled manner (i.e., $\lim_{x\to 0, y\to 1} \frac{f'(x)}{f'(y)} = 49$) with less lost in information provided with the variance of data. Using this function with $\theta = 10^{-7}$, we could get UAS of 92.3 and LAS of 90.9 on WSJ development set annotated with Stanford typed dependencies, which is slightly better than other transformation functions (see Fig. 2b). We obtain UAS of **92.0** and LAS of **90.6** on the WSJ test set with the same setting.

6.2 Dimensionality

The dimensionality of word vectors is determined by the number of singular vectors used in Eq. 1.

[7] http://nlp.cs.lth.se/software/treebank-converter/

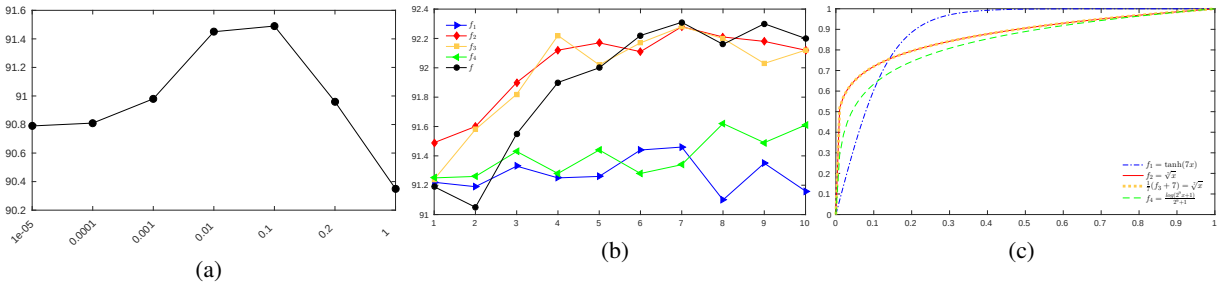


Figure 2: Unlabelled attachment score of parsing models trained with (**a**) the randomly generated word vectors, and (**b**) the systematically extracted word vectors using different transformation functions. The experiments are carried out with 50 dimensional word vectors. Parsing models are evaluated on the development set in Wall Street Journal annotated with Satnford typed dependencies (SD). f in (b) is the piecewise transformation function shown in Eq. 2. **c**: the transformation functions in each function set resulting in the best parsing models. The vertical axis shows the data in a probability co-occurrence matrix and the vertical axis shows their projection after transformation. For better visualization, the range of f_3 is scaled to $[0, 1]$.

High dimensional word vectors are expected to result in higher parsing accuracies. It is because they can capture more information from the original data, i.e., the Frobenius norm of the deference between the original matrix and its truncated estimation depends on the number of top singular vectors used for constructing the truncated matrix. This achievement, however, is at the cost of more computational resources a) to extract the word vectors, and b) to process the word vectors by parser. The most expensive step to extract the word vectors is the singular value decomposition of the transformed co-occurrence matrix. Using the randomized SVD method described by Tropp et al. (2009), the extraction of k top singular vectors of an $m \times n$ matrix requires $O(mn \log(k))$ floating point operations. It shows that the cost for having larger word vectors grows logarithmically with the number of dimensions.

The parsing performance is affected by the dimensionality of the word vectors, fed into the input layer of the neural network, in two ways: First, higher number of dimensions in the input layer lead to a larger weight matrix between the input layer and the hidden layer. Second, larger hidden layer is needed to capture the dependencies between the elements in the input layer. Given a set of word vectors with k dimensions connected to the hidden layer with h hidden units, the weight matrix between the input layer and the hidden layer grows with the scale of $O(kh)$, and the weight matrix between the hidden layer and the output layer grows with the scale of $O(h)$. For each input vector, the back-propagation algorithm passes the weight matrices three times per iteration 1) to forward each input vector through the network, 2) to back propagate the errors, generated by the inputs, and 3) to update the network parameters. So, each input vector needs $O(3(kh + h)))$ time to be processed by the algorithm. Given the trained model, the output signals are generated through only one forward pass.

Table 2 shows how high dimensional word vectors affect the parsing performance. In general, increasing the number of hidden units leads to a more accurate parsing model at a linear cost of parsing speed. Increasing the dimensionality of word vectors to 200 dimensions consistently increases the parsing accuracy at again the linear time of parsing speed. However, increasing both the dimensionality of word vectors and the size of the hidden layer leads to a quadratic decrease in parsing speed. The best results are obtained from the parsing model trained with 100-dimensional word vectors and 400 hidden units, resulting in the parsing accuracy of **93.0** UAS and **91.7** LAS on our test set, $+1.0$ UAS and $+1.1$ LAS improvement over what we obtained with 50 dimensional word vectors. It is obtained at the cost of 47% reduction in the parsing speed.

6.3 Comparison and Consistency

We evaluate the `RSV` word vectors on different types of dependency representations and different languages. Table 3 gives a comparison between `RSV` and different methods of word embedding with respect to their contributions to dependency parsing and the time required to generate word vectors for English. All the parsing models are trained with 400 hidden units and 100-dimensional word vectors extracted from English raw corpus described in Sec. 5. The word vec-

h→	200			300			400		
↓k	UAS	LAS	P	UAS	LAS	P	UAS	LAS	P
50	92.3	90.9	392	92.9	91.5	307	93.0	91.6	237
100	92.6	91.2	365	92.9	91.5	263	**93.1**	**91.8**	206
150	92.6	91.2	321	92.9	91.5	236	93.1	91.8	186
200	**92.7**	**91.3**	310	**93.1**	**91.7**	212	93.1	91.8	165
250	92.7	91.2	286	92.9	91.5	201	93.0	91.7	146
300	92.6	91.2	265	92.9	91.6	180	92.9	91.5	119
350	92.7	91.2	238	92.8	91.4	174	92.9	91.5	111
400	92.6	91.2	235	92.8	91.3	141	93.0	91.5	97

Table 2: The performance of parsing models trained with k-dimensional word vectors and h hidden units. The parsing accuracies (UAS, LAS) are for the development set. P: parsing speed (sentence/second).

Model	Time	SD		CD		UD	
		UAS	LAS	UAS	LAS	UAS	LAS
		p-val	p-val	p-val	p-val	p-val	p-val
CBOW	8741	93.0	91.5	93.4	92.6	88.0	85.4
		0.00	0.00	0.02	0.00	0.00	0.00
SGram	11113	93.0	91.6	93.4	92.5	87.4	84.9
		0.06	0.02	0.00	0.00	0.00	0.00
GloVe	3150	92.9	91.6	93.5	92.6	**88.4**	85.8
		0.02	0.02	0.04	0.06	0.54	0.38
HPCA	**2749**	92.1	90.8	92.5	91.7	86.6	84.0
		0.00	0.00	0.00	0.00	0.00	0.00
RSV	2859	**93.1**	**91.8**	**93.6**	**92.8**	**88.4**	**85.9**

Table 3: Performance of word embedding methods: Quality of word vectors are measured with respect to parsing models trained with them. The efficiency of models is measured with respect to the time (seconds) required to extract a set of word vectors. Parsing models are evaluated on our English development set; SGram: SkipGram, SD: Stanford typed Dependencies, CD: CoNLL Dependencies, UD: Universal Dependencies, and p-val: p-value of the null hypothesis: *RSV is no better than the word embedding method corresponding to each cell of the table.*

tors are extracted by a Linux machine running on 12 CPU cores. The free parameters of the word embedding methods, i.e., context type and context size, have been tuned on the development set and the best settings, resulting in the highest parsing accuracies, were then chosen for comparison. It leads us to asymmetric window of length 1 for RSV, GloVe and HPCA, and symmetric window of length 1 for word2vec models, CBOW and SkipGram. The GloVe word vectors are extracted by available implementation of GloVe (Pennington et al., 2014) running for 50 iterations. The HPCA word vectors are extracted by our implementation of the method (Lebret and Collobert, 2014). CBOW and SkipGram word vectors are extracted by available implementation of word2vec (Mikolov et al., 2013) running for 10 iterations, and a negative sampling value of 5. In order to show the validity of the results, we perform a bootstrap statistical significance test (Berg-Kirkpatrick et al., 2012) on the results obtained from each parsing experiment and RSV with the null hypothesis H_0: *RSV is no better than the model B*, where B can be any of the word embedding methods. The resulting p-values are reported together with the parsing accuracies.

The empirical results show that HPCA, RSV, and GloVe are ranked as fastest methods of word embedding in order of time. The reason why these methods are faster than word2vec is because they scan the corpus only one time and then store it as a co-occurrence matrix in memory. The reason why HPCA is faster than RSV is because HPCA stores the co-occurrence matrix as a sparse matrix but RSV stores it as a full matrix. This expense makes RSV more qualified than HPCA when they are used in the task of dependency parsing.

The results obtained from RSV word vectors are comparable and slightly better than other sets of word vectors. The difference between RSV and other methods is more clear when one looks at the difference between the labelled attachment scores. Apart from the parsing experiment with GloVe on the universal dependencies, the relatively small p-values reject our null hypothesis and confirms that RSV can result in more qualified word vectors for the task of dependency parsing. In addition to this, the constant superiority of the results obtained from RSV on different dependency styles is an evidence that the results are statistically significant, i.e., the victory of RSV is not due merely to chance. Among the methods of word embedding, we see that the results obtained from GloVe are more close to RSV, especially when they come with universal dependencies. We show in Sec. 8 how these methods are connected to each other.

Table 4 shows the results obtained from Stanford parser trained with RSV vectors and two other *greedy* transition-based dependency parsers MaltParser (Nivre et al., 2006) and Parsito (Straka et al., 2015). All the parsing models are trained with the arc-standard system on the corpus of universal dependencies. Par-St and Par-Sr refer to the results reported for Parsito trained with static oracle and search-based oracle. As shown, in all cases, the parsing models trained with Stanford parser and RSV (St-RSV) are more accurate than other parsing models. The superiority of the results obtained from St-RSV to Par-Sr shows the importance of word vectors in dependency parsing in comparison with adding more features to

the parser or performing the search-based oracle.

	Par-St	Par-Sr	Malt	St-RSV
	UAS LAS	UAS LAS	UAS LAS	UAS LAS
English	86.7 84.2	87.4 84.7	86.3 82.9	**87.6** **84.9**
Persian	83.8 80.2	84.5 81.1	80.8 77.2	**85.4** **82.4**
Swedish	85.3 81.4	85.9 82.3	84.7 80.3	**86.2** **82.5**

Table 4: Accuracy of dependency parsing. Par-St and Par-Sr refer to the `Parsito` models trained with static oracle and search-based oracle. St-RSV refers to the Stanford parser trained with `RSV` vectors.

The results obtained from Stanford parser and `UDPipe` (Straka et al., 2016) are summarized in Table 5. The results are reported for both predicted and gold POS tags. `UDPipe` sacrifice the greedy nature of `Parsito` through adding a beam search decoder to it. In general, one can argue that `UDPipe` adds the following items to the Stanford parser: 1) a search-based oracle, 2) a set of morphological features, and 3) a beam search decoder. The almost similar results obtained from both parsers for English show that a set of informative word vectors can be as influential as the three extra items added by `UDPipe`. However, the higher accuracies obtained from `UDPipe` for Swedish and Persian, and the fact that the training data for these languages are considerably smaller than English, show the importance of these extra items on the accuracy of parsing model when enough training data is not provided.

	Predicted tags		Gold tags	
	UDPipe	St-RSV	UDPipe	St-RSV
	UAS LAS	UAS LAS	UAS LAS	UAS LAS
English	84.0 80.2	**84.6** **80.9**	87.5 **85.0**	**87.6** 84.9
Swedish	**81.2** **77.0**	80.4 76.6	**86.2** **83.2**	**86.2** 82.5
Persian	**84.1** **79.7**	82.4 78.1	**86.3** **83.0**	85.4 82.4

Table 5: Accuracy of dependency parsing on the corpus of universal dependencies. St-RSV refers to the Stanford parser trained with `RSV` vectors.

7 Nature of Dimensions

In this section, we study the nature of dimensions formed by `RSV`. Starting from high-dimensional space formed by the transformed co-occurrence matrix $f(\mathbf{C})$, word similarities can be measured by a similarity matrix $K = f(\mathbf{C}^T)f(\mathbf{C})$ whose leading eigenvectors, corresponding to the leading right singular vectors of $f(\mathbf{C})$, form the `RSV` word vectors. It suggests that `RSV` dimensions measure a typical kind of word similarity on the basis of variability of word's contexts, since the eigenvectors of K account for the directions of largest variance in the word vectors defined by $f(\mathbf{C})$.

To assess the validity of this statement, we study the dimensions individually. For each dimension, we first project all unit-sized word vectors onto it and then sort the resulting data in ascending order to see if any syntactic or semantic regularities can be seen. Table 6 shows 10 words appearing in the head of ordered lists related to the first 10 dimensions. The dimensions are indexed according to their related singular vectors. The table shows that to some extent the dimensions match syntactic and semantic word categories discussed in linguistics. There is a direct relation between the indices and the variability of word's contexts. The regularities between the words appearing in highly variable contexts, mostly the high frequency words, are captured by the leading dimensions.

To a large extent, the first dimension accounts for the degree of variability of word's contexts. Lower numbers are given to words that appear in highly flexible contexts (i.e., high frequency words such as *as, but, in* and ...). Dimensions 2–5 are informative for determining syntactic categories such as adjectives, proper nouns, function words, and verbs. Dimensions 2 and 8 give lower numbers to proper names (interestingly, mostly last names in 2 and male first names in 8). Some kind of semantic regularity can also be seen in most dimensions. For example, adjectives in dimension 2 are mostly connected with society, nouns in dimension 6 denote humans, nouns in dimension 7 are abstract, and words in dimension 9 are mostly connected to affective emotions.

8 Related Work on Word Vectors

The two dominant approaches to creating word vectors (or word embeddings) are: 1) incremental methods that update a set of randomly initialized word vectors while scanning a corpus (Mikolov et al., 2013; Collobert et al., 2011), and 2) batch methods that extract a set of word vectors from a co-occurrence matrix (Pennington et al., 2014; Lebret and Collobert, 2014). Pennington et al. (2014) show that both approaches are closely related to

Dim	Top 10 words
1	. – , _ is as but in ... so
2	domestic religious civilian russian physical social iraqi japanese mexican scientific
3	mitchell reid evans allen lawrence palmer duncan russell bennett owen
4	. but in – the and or as at ,
5	's believes thinks asks wants replied tries says v. agrees
6	citizens politicians officials deputy businessmen lawmakers former elected lawyers politician
7	cooperation policy reforms policies funding reform approval compliance oversight assistance
8	geoff ron doug erik brendan kurt jeremy brad ronnie yuri
9	love feeling sense answer manner desire romantic emotional but ...
10	have were are but – _ . and may will

Table 6: Top 10 words projected on the top 10 dimensions

each other. Here, we elaborate the connections between RSV and HPCA (Lebret and Collobert, 2014) and GloVe (Pennington et al., 2014).

HPCA performs Hellinger transformation followed by principal component analysis on co-occurrence matrix $\mathbf{C}$ as below:

$$Y = SV^T \tag{3}$$

where Y is the matrix of word vecors, and S and V are matrices of top singular values and right singular vectors of $\sqrt[2]{\mathbf{C}}$. Since the word vectors are to be used by a neural network, Lebret and Collobert (2014) recommend to normalize them to avoid the saturation problem in the network weights (LeCun et al., 2012). Denoting $\tilde{Y}$ as the empirical mean of the column vectors in Y and $\sigma(Y)$ as their standard deviation, Eq. 4 suggested by Lebret and Collobert (2014) normalizes the elements of word vectors to have zero mean and a fixed standard deviation of $\lambda \leq 1$.

$$\Upsilon = \frac{\lambda(Y - \tilde{Y})}{\sigma(Y)} \tag{4}$$

$\tilde{Y}$ is $\mathbf{0}$ if one centres the column vectors in $\sqrt[2]{\mathbf{C}}$ around their mean *before* performing PCA. Substituting Eq. 3 into Eq. 4 and the facts that $\tilde{Y} = \mathbf{0}$ and $\sigma(Y) = \frac{1}{\sqrt{n-1}}S$, where n is the number of words, we reach Eq. 1.

In general, one can argue that RSV generalises the idea of Hellinger transformation used in HPCA through a set of more general transformation functions. Other differences between RSV and HPCA are in 1) how they form the co-occurrence matrix $\mathbf{C}$, and 2) when they centre the data. For each word w_i and each context word w_j, $\mathbf{C}_{i,j}$ in RSV is $p(w_j|w_i)$, but $p(w_i|w_j)$ in HPCA. In RSV, the column vectors of $f(\mathbf{C})$ are centred around their means before performing SVD, but in HPCA, the data are centred after performing PCA. In Sec. 6, we showed that these changes result in significant improvement in the quality of word vectors.

The connections between RSV and GloVe is as follows. GloVe extracts word vectors from a co-occurrence matrix transformed by logarithm function. Using a global regression model, Pennington et al. (2014) argue that linear directions of meanings is captured by the matrix of word vectors $\Upsilon_{n,k}$ with following property:

$$\Upsilon^T\Upsilon = \log(\mathbf{C}) + \mathbf{b1} \tag{5}$$

where, $\mathbf{C}_{n,n}$ is the co-occurrence matrix, $\mathbf{b}_{n,1}$ is a bias vector, and $\mathbf{1}_{1,n}$ is a vector of ones. Denoting Υ_i as ith column of Υ and assuming $\|\Upsilon_i\| = 1$ for $i = 1 \ldots n$, the left-hand side of Eq. 5 measures the cosine similarity between unit-sized word vectors Υ_i in a kernel space and the right-hand side is the corresponding kernel matrix. Using kernel principal component analysis (Schölkopf et al., 1998), a k-dimensional estimation of Υ in Eq. 5 is

$$\Upsilon = \sqrt{S}V^T \tag{6}$$

where S and V are the matrices of top singular values and singular vectors of K. Replacing the kernel matrix in Eq. 5 with the second degree polynomial kernel $K = f(\mathbf{C}^T)f(\mathbf{C})$, which measures the similarities on the basis of the column vectors defined by the co-occurrence matrix, the word vectors generated by Eq. 6 and Eq. 1 are distributed in the same directions but with different variances. It shows that the main difference between RSV and GloVe is in the kernel matrices they are using.

9 Conclusion

In this paper, we have proposed to form a set of word vectors from the right singular vectors of a co-occurrence matrix that is transformed by a 7th-root transformation function. It has been shown that the proposed method is closely related to previous methods of word embedding such as HPCA and GloVe. Our experiments on the task of dependency parsing show that the parsing models trained with our word vectors are more accurate than the parsing models trained with other popular methods of word embedding.

References

Abolfazl AleAhmad, Hadi Amiri, Ehsan Darrudi, Masoud Rahgozar, and Farhad Oroumchian. 2009. Hamshahri: A standard persian text collection. *Knowledge-Based Systems*, 22(5):382–387.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 995–1005, Stroudsburg, PA, USA. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Marie-Catherine De Marneffe and Christopher D Manning. 2010. Stanford typed dependencies manual (2008). *URL:* `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *16th Nordic Conference of Computational Linguistics*, pages 105–112. University of Tartu.

Rémi Lebret and Ronan Collobert. 2014. Word embeddings through hellinger pca. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 482–490, Gothenburg, Sweden, April. Association for Computational Linguistics.

Rémi Lebret and Ronan Collobert. 2015. Rehabilitation of count-based models for word vector representations. In *Computational Linguistics and Intelligent Text Processing*, pages 417–429. Springer.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics - Special issue on using large corpora*, 19(2):313 – 330, June.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319.

Mojgan Seraji. 2015. *Morphosyntactic Corpora and Tools for Persian*. Ph.D. thesis, Uppsala University.

Milan Straka, Jan Hajic, Jana Straková, and Jan Hajic jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*, pages 208–220.

Milan Straka, Jan Hajic, and Jana Strakov. 2016. Udpipe: Trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may. European Language Resources Association (ELRA).

A Tropp, N Halko, and PG Martinsson. 2009. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical report, Applied & Computational Mmathematics, California Institute of Technology.