Real-valued Syntactic Word Vectors

A. Basirat and J. Nivre

Department of Linguistics and Philology, Uppsala University

ARTICLE HISTORY
Compiled May 31, 2019

ABSTRACT
We introduce a word embedding method that generates a set of real-valued word vectors from a distributional semantic space. The semantic space is built with a set of context units (words) which are selected by an entropy-based feature selection approach with respect to the certainty involved in their contextual environments. We show that the most predictive context of a target word is its preceding word. An adaptive transformation function is also introduced that reshapes the data distribution to make it suitable for dimensionality reduction techniques. The final low-dimensional word vectors are formed by the singular vectors of a matrix of transformed data. We show that the resulting word vectors are as good as other sets of word vectors generated with popular word embedding methods.

1. Introduction

The distributional representation of words, known as word embeddings or word vectors, have shown great improvements on the performance of natural language processing tasks such as part-of-speech tagging (Collobert et al., 2011) and parsing (Chen & Manning, 2014; Dyer, Ballesteros, Ling, Matthews, & Smith, 2015). Words in these representations are modelled through real-valued feature vectors that capture global syntactic and semantic dependencies between words. These features provide for the application of machine learning techniques that can efficiently process continuous data with reasonable time and memory complexity (e.g., neural networks).

Real-valued Syntactic word Vectors (RSV) (Basirat & Nivre, 2017) is a method of word embedding that builds a set of word vectors from right singular vectors of a transformed co-occurrence matrix. `RSV` uses an nth root transformation function to reshape the distribution of the co-occurrence data. The co-occurrence matrix forms a semantic space whose dimensions correspond to a set of context units. The context units are the set of most frequent words preceding all words of interest. The transformation function normalizes the data distribution in the semantic space and makes the data more suitable for performing dimensionality reduction.

This paper addresses two problems with `RSV`: First, context units in `RSV` are chosen only on the basis of word frequencies and the model does not provide any way to use other informative features. `RSV` uses a limited number of frequent words as context units and completely ignores the less frequent words. Second, `RSV` does not provide any systematic solution to define the transformation function. It uses a predefined

Department of Linguistics and Philology, Uppsala University, Box 635, 751 26 UPPSALA, SWEDEN Email: {ali.basirat,joakim.nivre}@lingfil.uu.se

transformation function which is experimentally tuned with regard to the performance of word vectors on a task (e.g., dependency parsing). These problems are also seen in other word embedding methods such as `GloVe` (Pennington, Socher, & Manning, 2014), `word2vec` (Mikolov, Chen, Corrado, & Dean, 2013), and `HPCA` (Lebret & Collobert, 2014). In these methods, the context units are restricted to the most frequent words and the transformation function is tuned by their end users.

We propose two solutions to mitigate the aforementioned problems with `RSV`. In order to remedy the first problem, we add an entropy-based feature selection algorithm to `RSV` to find a set of more appropriate context units not only on the basis of word frequencies but also on the basis of the certainty involved in their contextual environment. The second problem is solved by an adaptive transformation which normalizes the data distribution in the semantic space formed by `RSV`. These changes make `RSV` a simple and adaptive model that is capable of automatically finding the best set of parameters with respect to a training data set.

We evaluate the word vectors with regard to their contributions to the dependency parsing task. Our experimental results show that the proposed solutions effectively improve the quality of the word vectors. We also compare `RSV` with other popular word embedding methods. The comparison is done with regard to a word similarity benchmark and different NLP tasks such as named-entity recognition, part-of-speech tagging, and dependency parsing. The results obtained by `RSV` are on par with the results obtained from other word embedding methods.

2. Word Vectors in Distributional Semantic Space

The word embedding methods can be divided into two major classes: 1) the methods that are developed in the area of distributional semantics (Basirat & Nivre, 2017; Landauer & Dumais, 1997; Lebret & Collobert, 2014; Lund & Burgess, 1996; Pennington et al., 2014; Sahlgren, 2006; Schütze, 1992) and 2) the methods that are developed in the area of language modeling (Collobert et al., 2011; Mikolov, Chen, et al., 2013). Levy and Goldberg (2014b) show that both classes are connected to each other. In both areas, a set of word vectors is generated through the application of a dimensionality reduction technique on a co-occurrence matrix, which is built either explicitly (Basirat & Nivre, 2017; Lebret & Collobert, 2014; Pennington et al., 2014) or implicitly (Collobert et al., 2011; Mikolov, Chen, et al., 2013; Sahlgren, 2006) while scanning a raw corpus. In this section, we review the previous work in the area of distributional semantics since `RSV` is categorized in this area.

A distributional semantic space is a finite dimensional vector space (or linear space) whose dimensions correspond to the contextual environment of words in a corpus. Word similarities in a distributional semantic space are reflected through the similarities between vectors associated with them. In other words, similar vectors are associated with similar words.

A distributional semantic space can be built in three main steps: First, each word is associated with a vector, called *context-word* vector. The elements of a context-word vector associated with a word are the frequencies of seeing the word in certain contexts. Context is a region in a text whose presence can be attributed to a set of textual units, called *context units.* The relationships between words and context units in certain contexts are measured by a function, called *local weighting function.* In the second step, a transformation function is applied on the context-word vectors in order to weight the importance of contexts in discriminating between words. We will refer to this function as a *global weighting function*. Finally, in the third step, a set of low dimensional word vectors are extracted from these transformed context-word vectors through the application of dimensionality reduction techniques.

The matrix formed by initial context-word vectors is called a *co-occurrence* matrix. Algorithm 1 gives a bird's-eye view on how a co-occurrence matrix is built. The algorithm takes three lists as input:

(1) a corpus $E = (e_1 \dots e_T)$ consisting of T words,
(2) a list of context units $\mathbb{C} = (c_1 \dots c_M)$, and
(3) a vocabulary $\mathcal{V} = (w_1 \dots w_N)$.

The algorithm returns an $M \times N$ co-occurrence matrix, $\mathcal{C}_{MN}$. The co-occurrence matrix is first initialised by zero. Then it is updated by repeatedly calling a *local weighting function* for each word $e_t \in E$ seen at time step t. The local weighting function measures the strength of relationship between word e_t and all context units in $\mathbb{C}$. Basically, it defines a contextual environment for each context unit in $\mathbb{C}$ and measures the extent to which e_t is in the contextual environment.

Algorithm 1 The co-occurrence matrix $\mathcal{C}_{MN}$ is computed for vocabulary $\mathcal{V}$ appearing in corpus E with respect to context units $\mathbb{C}$. The ith column of $\mathcal{C}$ $(\mathcal{C}_{:,i})$, which is the context-word vector associated with the ith word in $\mathcal{V}$, is updated for all words $e_t \in E$.

```
procedure Cooccurrence($E, \mathbb{C}, \mathcal{V}$)
    $(T, M, N) \leftarrow$ Length($E, \mathbb{C}, \mathcal{W}$)
    $\mathcal{C}_{MN} \leftarrow \mathbf{0}$
    for $t \leftarrow 1$ to $T$ do
        $i \leftarrow$ Lookup($e_t, \mathcal{V}$)
        $\mathcal{C}_{:,i} \leftarrow \mathcal{C}_{:,i}$ + Local-Weight($t, \mathbb{C}, E$)
    end for
    return $\mathcal{C}$
end procedure
```

Different types of context units and local weighting functions have been used in the literature, resulting in two major types of contextual environments:

(1) window-based context
(2) syntax-based context

A window-based context is formed by all words in a sequence (or window) of words associated with a context unit. A syntax-based context is formed by words that are in certain syntactic relations with a context unit. The relative position of words is either completely ignored by these types of context or in some cases they are modelled by some weights associated with words with respect to their distance to the context unit.

Latent semantic analysis (LSA) (Landauer & Dumais, 1997) and hyperspace analogue to language (HAL) (Lund & Burgess, 1996) are among the many methods that use the window-based context. The context units in LSA are a list of documents whose union forms the corpus E. The contextual environment of each element in $\mathcal{C}_{LSA}$ is all words in the corresponding document, a window with the length of the document. For each word e_t, the local weighting function in LSA returns a one-hot vector with all elements equal to zero except for one which indicates the document that e_t belongs to. Schütze (1992) and HAL define the context units as a set of words and the contextual environment of the context units as their surrounding words in a window with a certain length. For each word e_t, the local weighting function in HAL returns a real-valued vector whose elements are proportional to the number of words between e_t and each of the context units (words). Fine-grained grammatical tags such as stem, lemma, and part-of-speech tag are also used as context units. Kiela and Clark (2014) define context units as the lemmas of words in conjunction with the CCG supertags associated with them.

The syntax-based context is usually built over the dependency relations between words. Padó and Lapata (2007) define the contextual environment of a word in a sentence as all words in the sentence which are in a dependency relation with the target word. Although Padó and Lapata (2007) show that the dependency context can result in higher accuracies in certain tasks, Kiela and Clark (2014) argue that better results can be obtained from window-based context if the vectors are extracted from a fairly large corpus with a small window size. In Sec. 3.1, we introduce an entropy-based measurement to evaluate the quality of different types of contextual environments.

Once a co-occurrence matrix is computed, it is normalized by a *global weighting function* (e.g., pointwise mutual information (PMI) (Church & Hanks, 1990), term frequency-inverse document frequency (TF-IDF) (Salton & Buckley, 1988), term frequency-inverse corpus frequency (TF-ICF) (Reed et al., 2006), and relative frequencies (Lebret & Collobert, 2014)). Regardless of the type of local and global weighting functions, the set of context-word vectors extracted from a corpus can be highly massed around zero. This is because of the Zipfian distribution of words that skews the vector's mass toward less frequent words. The dimensionality of the context-word vectors and their Zipfian distribution limit the practical usage of the vectors. `GloVe` (Pennington et al., 2014), `HPCA` (Lebret & Collobert, 2014), and `RSV` (Basirat & Nivre, 2017) mitigate this problem by performing a power transformation function on the elements of the co-occurrence matrix. However, none of these methods provide a systematic way to set the parameters of the transformation function (e.g., power value) and leave them as user defined parameters. In Sec. 3.2, we introduce a transformation function whose parameters are automatically adjusted based on the data distribution along each dimension of the semantic space.

The problem of dimensionality is mitigated by techniques of dimensionality reduction. Principal component analysis is a standard technique of dimensionality reduction that is widely used for this purpose. Mixture models (Blei, Ng, & Jordan, 2003; Hofmann, 1999), and non-linear methods (Hinton & Salakhutdinov, 2006; Roweis & Saul, 2000) are among the other approaches that have been used for reducing the dimensionality of a semantic space. In this paper, the low-dimensional word vectors are formed by a linear transformation of the right singular vectors of the transformed co-occurrence matrix (see Sec. 5.4).

3. Real-Valued Syntactic Word Vectors (RSV)

We adopt the three main steps mentioned in Sec. 2 to build a set of word vectors for syntactic analysis of languages. In the following sections, we elaborate each of these steps in details.

3.1. Context-Word Vectors

Context-word vectors in our model are built on a set of context units which are basically lower-cased word forms. Algorithm 2 shows how a local weighting function, IN-CONTEXT, updates a context-word vector. Function IN-CONTEXT measures the extent to which the word at position t in corpus E belongs to each of the contextual environments built by context units $\mathcal{C}$.

Algorithm 2 The context-word vector C is computed for the word seen at position t in the corpus E. The context-word vector is built with respect to the list of context units $\mathcal{C} = (c_1 \dots c_M)$.

```
procedure LOCAL-WEIGHT($t, \mathcal{C}, E$)
    $M \leftarrow$ LENGTH($\mathcal{C}$)
    $C_{M1} \leftarrow \mathbf{0}$
    for $m \leftarrow 1$ to $M$ do
        $C_m \leftarrow$ IN-CONTEXT($t, \mathcal{C}(m), E$)
    end for
    return $C$
end procedure
```

Depending on how IN-CONTEXT is implemented, several types of contextual environment can be designed. We have considered four types of contextual environments, window-based symmetric/asymmetric and syntax-based symmetric/asymmetric. In the window-based symmetric context, for each context unit $c_m = \mathcal{C}(m)$, IN-CONTEXT returns $\sum_{i=1}^{n(c_m)} \frac{1}{l_i}$, where l_i is the positional distance between t and the ith occurrence of c_m out of total $n(c_m)$ occurrences before and after t in E. It is quite common to define a threshold value $n \in \mathbf{N}$ on the values of l in order to filter out those occurrences of c_m which are far from t. This threshold value is referred to as *window size* in the literature. In the window-based asymmetric context we use a similar function but instead of scanning both sides of t we only scan the left side of t.

Syntax-based contexts are defined with respect to the dependency relations between context units and words. In the syntax-based asymmetric context, l_i is the length of dependency path between context unit c_m and t if c_m is in the list of ancestors of word e_t. We factor out all context units that are not among the ancestors of e_t or whose path length from t exceeds the threshold value of n ($n < l_i$). In the syntax-based symmetric context, we look at both ancestors and descendants of e_t.

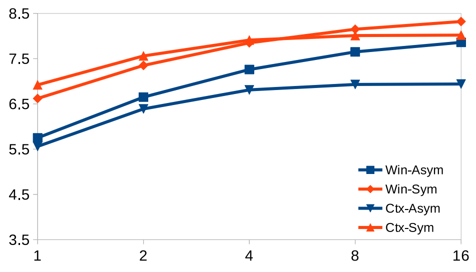
We propose to evaluate different types of contextual environments by the average value of the conditional entropy of seeing words in the contextual environments (see Eq. 1).

$$H(\mathbfcal{V}|\mathbfcal{C}) = \sum_{m=1}^{M} p(c_m) H(\mathbfcal{V}|\mathbfcal{C} = c_m) \tag{1}$$

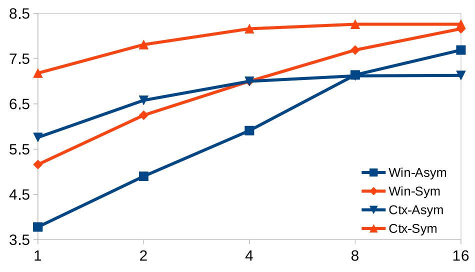
The entropy measures the average amount of uncertainty involved in describing a word through each of the context units. In other words, it measures the extent to which a word can be predicted from a context unit.

Fig. 1a shows the average values of conditional entropies for different types of contextual environment with respect to the different threshold values of n. In general the contextual environments defined by asymmetric local weighting functions with small values of n are more certain in describing words that appear in their domains. The best contextual environment is defined by asymmetric dependency weighting function with $n = 1$. It shows that among the other types of contextual environments, the immediate parent of a word in a dependency tree is the best descriptor of the word.

This conclusion is compatible with what Padó and Lapata (2007) show in their experiments. It also shows that asymmetric contexts are more certain than symmetric contexts. This is compatible with the experimental conclusion made by Lebret and Collobert (2015).



(a)



(b)

Figure 1. Vertical axis: the average conditional entropy of seeing a word given a context unit (a) before and (b) after context selection. Horizontal axis: the threshold value of n. Win: Window-based, Ctx: Context-based, Asym: Asymmetric, Sym: Symmetric

However, if the corpus size is large, it might be computationally expensive to first parse a corpus and then build context-word vectors from the parsed corpus. A solution to this problem is to build the vectors from window-based contextual environments whose context units are restricted to a subset of *high frequent words* which are more *certain* about describing words in their context. In order to find a proper criterion to choose these sorts of words we need to study the relationship between word frequencies and the amount of uncertainty involved in their contextual environments.

For each context unit $c \in \mathfrak{C}$, Fig. 2a shows how the marginal frequency of c is related to the uncertainty involved in its contextual environment. The uncertainty is measured as the conditional entropy of seeing words in the contextual environments of c (see Eq. 2).

$$H(\mathfrak{V}|\mathfrak{C}=c) = -\sum_{n=1}^{N} p(w_n|c)\log p(w_n|c) \tag{2}$$

Fig. 2c shows the histogram of the two-dimensional random variable consisting of the marginal frequency of context units and the conditional entropy of seeing words in their contextual environments. The figures show that there is almost no correlation between the two variables, word frequencies and conditional entropies. The small value of correlation between the variables, 0.04, confirms that word frequencies are not sufficient to choose a set of context units with the mentioned properties since this parameter does not provide enough information about the contextual uncertainty. The value of correlation can be increased to 0.44 if we logarithmically scale the word frequencies (Fig. 2b and Fig. 2d). In this case too we do not see a strong relationship between the variables. This suggests the necessity for performing a feature selection that takes the two parameters into consideration.

The following weighting function gives higher weights to the context units with the desired properties discussed above

$$\mathcal{W}(c) = p(c)\exp\Big(-\frac{H(\mathfrak{V}|\mathfrak{C}=c) - H(\mathfrak{V}|\mathfrak{C})}{2\sigma^2}\Big) \tag{3}$$

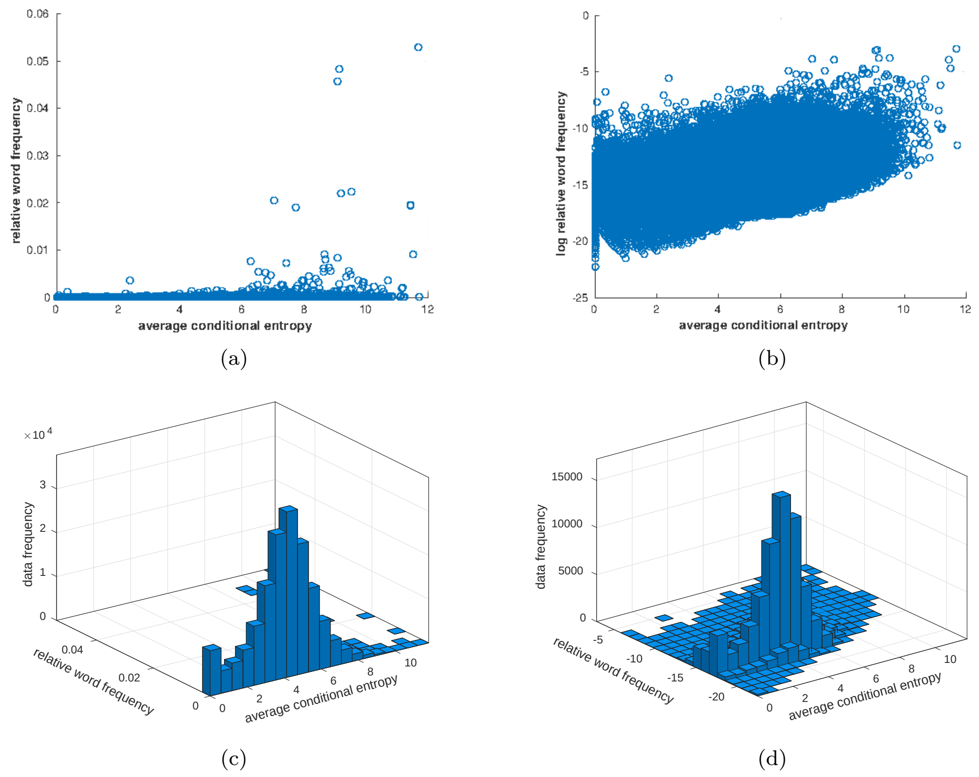
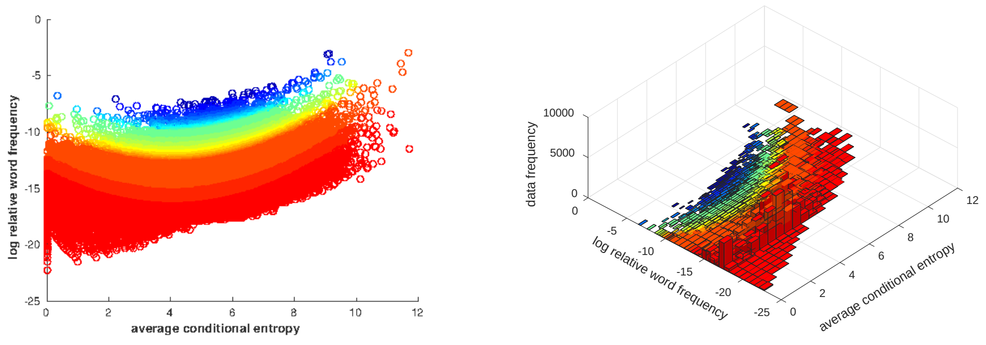


Figure 2. Top: the conditional entropy of contextual environments of words (horizontal axis) versus (a) relative word frequency, (b) logarithm of relative word frequencies. Bottom: the two-dimensional histograms of the conditional entropy of contextual environments of words and (c) the relative frequency of words, (d) the logarithm of relative word frequencies.

where $c \in \mathfrak{C}$ is a context unit with the marginal probability of $p(c)$; $H(\boldsymbol{\mathfrak{V}}|\boldsymbol{\mathfrak{C}})$ and σ are the mean and standard deviation of values of $H(\boldsymbol{\mathfrak{V}}|\boldsymbol{\mathfrak{C}} = c_i)$ for $i = 1 \dots M$ (see Eq. 1 and Eq. 2). Using Eq. 3, a set of context-word vectors can be built from the D best context units weighted by $\mathcal{W}$. The parameter D is a user defined parameter that controls the dimensionality of context-word vectors. Fig. 3 shows how the weighting function chooses high frequent words with low contextual entropies among all words. Depending on the value of D, context selection filters out words in different ranges of frequency including many of the most frequent words. Our experiment on the Wall Street Journal corpus (Marcus, Santorini, & Marcinkiewicz, 1993) shows that around 60% of the selected words overlap with the words that are more likely to appear as a parent node in a dependency tree. It shows that the context selection can be used as a replacement for parsing since it could capture a large portion of the context units which could be determined through parsing the corpus.

In general, theoretically and practically, smaller values of D are more desirable since it results in a smaller semantic space that can be analysed efficiently. However, a small value of D drastically skews the data distribution in the resulting semantic space toward zero. It is because of the presence of many zeros in the context-word vectors which in turn is due to the fact that the weighting function gives higher weights to the context units with smaller values of the conditional entropy, i.e., context units whose contextual environment is more predictable. So the context-word vectors formed by these context units are even more sparse than those that are formed with



(a) (b)

Figure 3. (a): the selected words after context selection with different values of D corresponding to the colour spectrum starting with $D = 20$, dark blue, to $D = 200\,000$, dark red. Horizontal axis: conditional entropy of contextual environments of words; Vertical axis: the logarithm of word frequencies. (b): the histogram of the selected words after context selection.

frequency-based context selection, where context units are selected only with respect to their frequencies. The skewness in the data distribution makes the application of dimensionality reduction techniques unreliable. In the next section, we introduce a transformation function to mitigate this problem.

Fig. 1b shows the average value of conditional entropy of seeing words in the contextual environment of context units selected by Eq. 3. In comparison with Fig. 1a, an asymmetric window-based context can be even better than a dependency context if we eliminate the less informative context units. It can also be seen that neither of the contextual environments formed by dependency trees are affected by the context selection. It shows that the weighting function preserves the context units that are defined by a dependency weighting function.

3.2. Transformation

We propose to transform the raw context-word vectors in two steps. First we apply a global weighting function on the elements of context-word vectors. It can be any of the popular methods used in the literature. This transformation is helpful to reduce the disproportionate contribution of the most frequent context units in the context-word vectors. However, it does not affect the problem with the data skewness mentioned in Sec. 3.1.

In the second step we normalize the data distribution along each dimension of the context-word vectors. The normalization is performed through a transformation function that maximizes the mean value of the entropy of data projected onto each dimension of the context-word vectors, i.e., the elements of each row in the co-occurrence matrix. The commonly used transformation functions for this purpose are monotonically increasing concave functions that magnify small numbers in their domain while preserving the given order (e.g., the logarithm function, the hyperbolic tangent, the power transformation, and the Box-Cox transformation). Given a set of transformation functions $f(x;\theta)$ with parameter θ, we define the best function as one with optimal parameter θ as follow:

$$\theta = \arg\max_{\theta} \sum_{d=1}^{D} p(c_d) H(f(\mathbf{X}_d; \theta)) \tag{4}$$

where d is the dimension in the semantic space corresponding to the context unit c_d, and $\mathbf{X}_d$ is the random variable taking the value of data points in the semantic space projected on the dimension d, i.e., the row in the co-occurrence matrix corresponding to c_d.

The unknown distribution of the data along the different dimensions of the distributional semantic space makes it difficult to analytically optimize the functional in Eq. 4, hence the need for using iterative optimization techniques such as genetic algorithms or simulated annealing. Since the computation of the objective function can be expensive for a large number of dimensions, D, we propose to compute the function for a small number of dimensions, say 100, which are randomly selected with respect to the frequency of seeing their corresponding context units, $p(c_d)$.

3.3. Dimensionality Reduction

The final low dimensional word vectors are built from the transformed context-word vectors in two steps. First the vectors are centred around their mean as below:

$$\mathbf{C}_{DN} = f(\mathcal{C}; \theta) - \mathbf{E}[f(\mathcal{C}; \theta)] \tag{5}$$

The final k-dimensional word vectors are then formed by right singular vectors of matrix $\mathbf{C}_{DN}$ as below:

$$\Upsilon = \lambda\sqrt{N}\mathbf{V}_{Nk}^{T} \tag{6}$$

where N is the number of words, λ is a constant scaling factor that controls the standard deviation of vectors, and $\mathbf{V}_{Nk}^{T}$ is the matrix formed by the top k right singular vectors of $\mathbf{C}$. Each row of Υ is associated with a word in vocabulary $\mathcal{V}$.

The final word vectors are centred around zero and have an almost diagonal covariance matrix $\lambda\mathbf{I}$. The former property is because of the mean subtraction step and the latter property is because the column vectors in $\mathbf{V_{Nk}}$ are almost orthogonal to each other. The near orthogonality is because the number of singular vectors is smaller than the total number of dimensions $k \ll D$ in the co-occurrence matrix. These properties make the vectors suitable to be used by the back-propagation algorithm as argued by LeCun, Bottou, Orr, and Müller (2012). In the case that word vectors are used by a neural network it is important to tune the parameter λ with respect to the network architecture. A proper value of λ prevents saturation in the hidden layers of the network and consequently increases the convergence speed of the network. In Sec. 5.2, we show how this parameter can be tuned empirically based on the contribution of the word vectors to a parsing task.

4. Connection with Other Methods

In this section, we study how `RSV` is connected with other word embedding methods including `HAL` (Lund & Burgess, 1996), `RI` (Sahlgren, 2006), `HPCA` (Lebret & Collobert, 2014) and `GloVe` (Pennington et al., 2014). The reason for choosing these methods is that all of them are based on the general idea of word embedding developed in the area of distributional semantics. The connection between the methods developed in the area of distributional semantics and the methods developed in the area of language modeling (e.g., `SENNA` (Collobert et al., 2011) and `word2vec` (Mikolov, Chen, et al., 2013)) is studied by Levy and Goldberg (2014b).

In general, `RSV` follows the same architecture as the other word embedding methods developed in the area of distributional semantics. The main contribution of `RSV` is the well-defined feature selection approach and the transformation step it performs on the co-occurrence data. `RSV` uses an entropy-based feature selection to find the most informative context units but the other methods rely on the marginal frequencies of context units. Unlike the other methods that use fixed transformation functions, `RSV` uses an adaptive transformation function. In the remainder of this section, we study the relationship between `RSV` and the other word embedding methods.

Context-word vectors in `HAL` are built from word co-occurrences which are weighted with a windows-based symmetric local weighting function (see Sec. 3.1). Lund and Burgess (1996) state that the co-occurrence matrix formed by these vectors can be fairly well estimated by a small number of its principal components. The low dimensional word vectors in `HAL` are extracted from the co-occurrence matrix through performing a principal component analysis (PCA). `HPCA` generalizes this approach. In `HPCA`, Lebret and Collobert (2014) propose to transform the elements of the co-occurrence matrix by a Hellinger transformation before performing PCA. Given a probability co-occurrence matrix C, the low dimensional vectors in `HPCA` are generated by:

$$Y = U^T\sqrt{C} \tag{7}$$

or equivalently by

$$Y = SV^T \tag{8}$$

where Y is the matrix of low-dimensional data, USV^T is the singular value decomposition of $\sqrt{C}$. Denoting $\bar{Y}$ as the empirical mean of the column vectors in Y and $\sigma(Y)$ as their standard deviation, Eq. 9 suggested by Lebret and Collobert (2014) normalizes the elements of word vectors to have zero mean and a fixed standard deviation of $\lambda \leq 1$.

$$\Upsilon = \frac{\lambda(Y - \bar{Y})}{\sigma(Y)} \tag{9}$$

In addition to these steps, `RSV` performs a context selection before building the co-occurrence matrix. The context selection takes the marginal word frequencies and the amount of uncertainty in the word's contexts into consideration. The Hellinger transformation in `HPCA` is further generalized in `RSV`. `RSV` uses a general transformation function whose parameters are adjusted with respect to the data distribution. In `RSV`, the transformed context-word vectors are centred around their mean before performing the singular value decomposition. This is as opposed to HPCA which performs the data centring after dimensionality reduction. In other words, HPCA performs PCA on non-centred data. This makes the results of PCA unreliable especially for the dominant principal components. If HPCA centres the column vectors in $\sqrt{C}$ before performing PCA then $\bar{Y}$ in Eq. 9 is $\mathbf{0}$. Substituting Eq. 8 into Eq. 9 and the facts that $\bar{Y} = \mathbf{0}$ and $\sigma(Y) = \frac{1}{\sqrt{N}}S$, where N is the number of words, we reach Eq. 6.

Now, we turn our attention on the relationship between RSV and RI. Context selection in RI is on the basis of word frequencies. RI filters out the highly frequent context units, i.e., the function words that contain the grammatical information, and keeps the frequent content words. However, as we mentioned earlier, context selection in RSV is on the basis of word frequency and the amount of entropy involved in contextual environments. RSV filters out the less frequent context units whose contextual environment involves high entropy. The context units filtered by this approach can be both function words and content words. Another difference between RSV and RI is in the method of dimensionality reduction. Sahlgren (2006) proposes to reduce the dimensionality of the co-occurrence matrix through random projection (Achlioptas, 2001). Random projection is based on the Johnson-Lindenstrauss lemma stating that any point in a high-dimensional space can be projected to a lower-dimensional space without pointwise distance distortion between the points. It uses a random projection matrix whose unit length rows, forming the lower-dimensional space, are *almost* orthogonal to each other. RI employs this idea of *near orthogonality* and associates each context unit (word) in the language with a unit length random vector. The local weighting function in RI returns the random vector associated with the context unit whose contextual environment includes the word of interest, e_t. The final word vectors are gradually computed through accumulating the random vectors while scanning the corpus. This is equivalent with multiplying a co-occurrence matrix with a random projection matrix. In RSV, we associate a set of context units with *fully orthogonal* vectors and build a co-occurrence matrix from those vectors. The low dimensional word vectors are then extracted from the singular vectors of this matrix which is affected by a transformation function.

RSV and GloVe (Pennington et al., 2014) are related to each other as follows. The context selection in GloVe is on the basis of word frequencies. GloVe filters out the less frequent context units and preserves the most frequent context units. However, the entropy-based context selection in RSV may preserve the less frequent context units with small amount of entropy in their contextual environment. It may also filter out the frequent context units with large amount of entropy in their contextual environment. The other difference between GloVe and RSV is the transformation function they apply on a co-occurrence matrix. Word-vectors in GloVe are extracted from a logarithmically transformed co-occurrence matrix. However, RSV uses an adaptive power transformation to generate the word vectors. Using a global log-bilinear regression model, Pennington et al. (2014) argue that the linear directions of meanings is captured by the matrix of word vectors $\Upsilon_{N,k}$ with the following property:

$$\Upsilon^T \Upsilon = \log(C) + \mathbf{b1} \tag{10}$$

where, C_{NN} is the co-occurrence matrix, $\mathbf{b}_{N1}$ is a bias vector, and $\mathbf{1}_{1N}$ is a vector of ones. Denoting Υ_i as the ith column of Υ and assuming $\|\Upsilon_i\| = 1$ for $i = 1 \dots N$, the left-hand side of Eq. 10 measures the cosine similarity between the 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, Smola, & Müller, 1998), a k-dimensional estimation of Υ in Eq. 10 is

$$\Upsilon = \sqrt{S}V^T \tag{11}$$

where S and V are the matrices of top singular values and singular vectors of K. Replacing the kernel matrix in Eq. 10 with the second degree polynomial kernel $K = \mathbf{C}^T\mathbf{C}$, where $\mathbf{C}$ is the matrix defined in Eq. 5, the word vectors generated by Eq. 11 and Eq. 6 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 to extract the word vectors.

5. Experiments

Our experiments are organized in the following way: First we study the effect of `RSV` parameters (Sec. 5.2) on the dependency parsing task. Then in Sec. 5.3, we study the effect of the entropy-based feature selection on word vectors. In Sec. 5.4, we study the effect of the dimensionality of word vectors on the parsing. All of these experiments are run on our development sets. In Sec. 5.5, we report our final results on our test sets and compare them with other methods of parsing. Moreover, we compare `RSV` with other popular word embedding methods (Sec. 5.6). The comparison is done with regard to the efficiency and effectiveness of the word embeddings. Different evaluation metrics are used to compare the effectiveness of the word embeddings.

5.1. Experimental Settings

We use the Stanford dependency parser (Chen & Manning, 2014) for our parsing experiments. The parser is an arc-standard system (Nivre, 2004) with a three-layered feed-forward neural-network as its classifier which is trained by the standard back-propagation algorithm. The parsing models are trained on two datasets, Wall Street Journal (WSJ) (Marcus et al., 1993) annotated with Stanford basic dependencies (de Marneffe & Manning, 2008), and the English part of the corpus of Universal Dependencies (UD) version 1.2 (Nivre et al., 2016). 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, MacCartney, & Manning, 2006) is used for converting constituency trees to dependency trees. We report the parsing accuracies for both gold and predicted part-of-speech tags. The predicted part-of-speech tags in both corpora are assigned through ten-way jackknifing on the training sets. WSJ is tagged by the Stanford part-of-speech tagger (Toutanova, Klein, Manning, & Singer, 2003) and the universal dependency corpus is tagged by UDPipe (Straka, Hajic, & Straková, 2016). The stack-LSTM parser (Dyer et al., 2015) is used in one of our experiments together with the Stanford parser to show the sensitivity of the parameter λ in Eq. 6.

The comparison between `RSV` and other word embedding methods is done with regard to the contribution of the word vectors in a word similarity benchmark, the part-of-speech tagging, named entity recognition and dependency parsing. We use the word similarity benchmark introduced by Faruqui and Dyer (2014). The benchmark evaluates a set of word vectors in 13 different word similarity benchmarks. Each benchmark contains pairs of English words associated with their similarity rankings. The tool reports the correlation between the similarity rankings provided by the word similarity benchmark and the cosine similarity between the word vectors. In order to provide an overall view over the performance of word vectors, we report the average of the correlations obtained from all word similarity benchmarks.

We use the `veceval` tool (Nayak, Angeli, & Manning, 2016) for the part-of-speech tagging and the named-entity recognition. `veceval` is a word embedding evaluation framework that measures the contribution of a set of word embeddings to different downstream standard NLP tasks. `veceval` uses a two-layer neural network followed by a softmax layer for part-of-speech tagging and named-entity recognition. The words' labels are predicted from the embeddings of their surroundings. For each word in a sentence, the word embeddings of its surrounding words are concatenated and fed to the neural network. The part-of-speech tagging models are trained and tested on WSJ with the following split: sections 00–18 are used as training data, and sections 19–21 are used as test data. The named-entity recognition models are trained and tested on the data provided by the CoNLL-2003 shared task (Tjong Kim Sang & De Meulder, 2003). The shared task provides two test sets, *testa* and *testb*. We use the *testa* dataset as the test set.

The Stanford parser (Chen & Manning, 2014) is used for the parsing comparison. We use the WSJ data, prepared for parsing, for training and evaluating the parser. The parsing models are evaluated on the development set of WSJ (i.e. section 22). We compare the F_1-score of the labelled and unlabelled attachment scores.

The word vectors are extracted from the English Linguistic Data Consortium (LDC) tokenized with the Stanford tokenizer.[1] All numbers in the corpus are replaced with a special token <num> and the uppercase letters are converted to their corresponding lowercase forms. The vectors are extracted for the tokens with at least 100 times frequency of occurrence in the corpus, resulting in around 209 360 unique tokens. We set the functional f in Eq. 4 to $f(x;\theta) = x^{\theta}$ and search for the optimal value of θ using simulated annealing. The parameter θ is restricted to the interval $(0,1)$, i.e., $\theta \in (0,1)$.

5.2. `RSV` Parameters

`RSV` has three parameters that need to be set beforehand:

(1) the scaling parameter λ used in Eq. 6,
(2) the global weighting function (see Sec. 3.2), and
(3) the number of context units D used in context selection (see Sec. 3.1).

In order to show the effect of the parameter λ, we train the Stanford parser (Chen & Manning, 2014) and the stack-LSTM parser (Dyer et al., 2015) with different sets of word vectors randomly generated with normal distribution with zero mean and different covariance matrices $\sigma\mathbf{I}$, where $\sigma \in (0,1]$. Fig. 4 shows the parsing accuracies obtained from these vectors. The parsing models are trained and evaluated on the WSJ. The variations in the accuracies show the sensitivity of the neural classifiers to the data variance. The difference between the two optimal values, $\sigma = 0.1$ for the Stanford parser and $\sigma = 0.01$ for the stack-LSTM parser, shows that the best value of standard deviation varies with respect to the target tool.

We study the effect of the two other parameters on the quality of word vectors using different numbers of context units ranging from 50 to $200,000$ and the context weighting functions listed in Table 1. Fig. 5 shows how the functional in Eq. 4 changes with respect to each of these weighting functions. The values of the functional strictly

[1]`https://www.ldc.upenn.edu/`

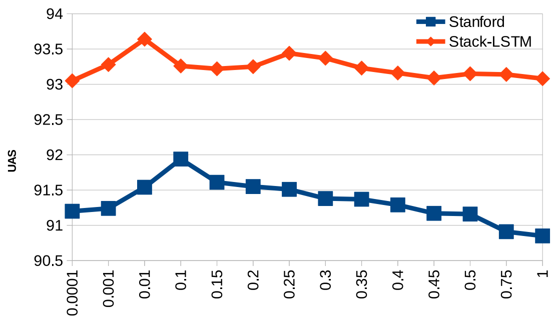


Figure 4. Vertical axis: unlabelled attachment scores obtained from parsing models trained with randomly initialized word vectors and gold part-of-speech tags. Horizontal axis: variance of the randomly generated vectors.

depends on the context weighting function. Among these weighting functions, *pcw* is the same as what Lebret and Collobert (2014) use with the Hellinger transformation. The figure shows that the optimal power value for this function is around 0.2 which is far from the Hellinger transformation (i.e., $\theta = 0.5$).

Name	Definition	
raw	$\mathcal{C}_{i,j}$	The raw co-occurrence frequency
pwc	$\frac{\mathcal{C}_{i,j}}{\sum_{l=1}^{N} \mathcal{C}_{i,l}}$	The relative frequency of seeing w_j given w_i
pcw	$\frac{\mathcal{C}_{i,j}}{\sum_{k=1}^{M} \mathcal{C}_{k,j}}$	The relative frequency of seeing w_i given w_j
pmi	$\max(0, \log(\frac{\mathcal{C}_{ij} \sum_{k=1}^{M} \sum_{l=1}^{N} \mathcal{C}_{k,l}}{(\sum_{k=1}^{M} \mathcal{C}_{k,j})(\sum_{l=1}^{N} \mathcal{C}_{i,l})}))$	The pointwise mutual information between w_j and w_i
tf-idf	$\log(\mathcal{C}_{i,j}) \log(\frac{N}{\sum_{l=1}^{N} \text{sign}(\mathcal{C}_{i,l})})$	Term frequency-inverse document frequency
tf-icf	$\log(\mathcal{C}_{i,j}) \log(\frac{\sum_{k=1}^{M} \sum_{l=1}^{N} \mathcal{C}_{k,l}}{\sum_{l=1}^{N} \mathcal{C}_{i,l}})$	Term frequency-inverse corpus frequency

Table 1. The list of context weighting functions applied on the co-occurrence matrix $\mathcal{C}_{M,N}$. $\mathcal{C}_{i,j}$ is the frequency of seeing word w_j in the contextual environment of context unit c_i. M is the number of context units, and N is the number of words.

Fig. 6 shows the optimal values of θ with respect to both global weighting functions and number of context units. The dimensionality is not a decisive factor in determining the optimal value of θ. This observation shows that the optimal value of θ can be computed from a small subset of dimensions. It confirms the solution proposed in Sec. 3.2 in response to the problem of efficiency, where θ is estimated from a set of randomly selected context units.

Fig. 7 is the histogram of the elements of the co-occurrence matrices before and after performing the transformation. Unsurprisingly, the power transformation function has no effect on the zero values of the co-occurrence matrices. The zero values are related to the *invalid contextual environments* that never occur in the raw corpus. The main effect of the transformation function is on the non-zero values in the co-occurrence matrix, which are related to the *valid contextual environments* that occur in the raw

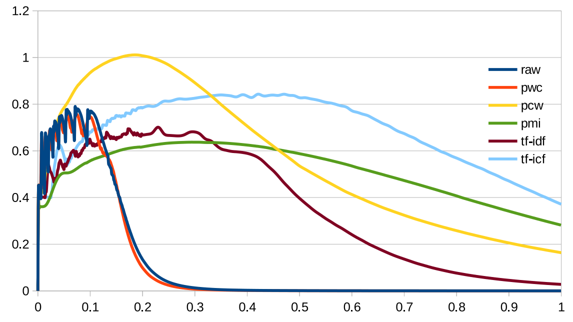


Figure 5. Behaviour of the functional in Eq. 4 (vertical axis) with $f = \sqrt[\theta]{x}$ and $D = 1000$ with respect to different values of $\theta \in (0, 1]$ (horizontal axis).

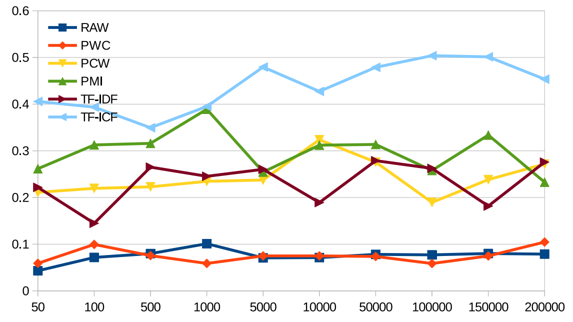


Figure 6. Optimal values of θ (vertical axis) maximizing the functional in Eq. 4 for English with respect to different values of D (horizontal axis) and different global weighting functions.

corpus. The transformation makes a clear distinction between zero and non-zero values in the co-occurrence matrix. In other words, for each word, the transformation function makes a clear distinction between the contextual environment that the word can appear in and those that the word never appear in. This distinction between the valid and invalid contextual environments is similar to the way that `word2vec` (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013) distinguishes between the valid and invalid contexts using logistic regression with negative sampling. The logistic regression in `word2vec` distinguishes between the valid contextual environments, positive samples seen in the training corpus, and the invalid contextual environments, negative samples that are randomly generated by the training algorithm. The randomly generated negative samples are not expected to be seen in the training corpus. In addition to making a distinction between the valid and invalid contextual environments, the transformation normalizes the distribution of the non-zero values. This is desirable for most of the dimensionality reduction methods such as principal component analysis, presuming that the data have normal distribution. Although PCA does not make any assumption about the data distribution, many of the interesting properties of PCA come from the data with normal distribution (Jolliffe, 2002).

Fig. 8 shows the accuracy of parsing models trained with the word vectors generated with different numbers of context units and global weighting functions. Apart from *pcw*, all the other functions result in almost the same accuracies. The fact that the word vectors obtained from *raw* frequencies are as good as other word vectors obtained from more complex weighting functions shows that the problems with disproportionate

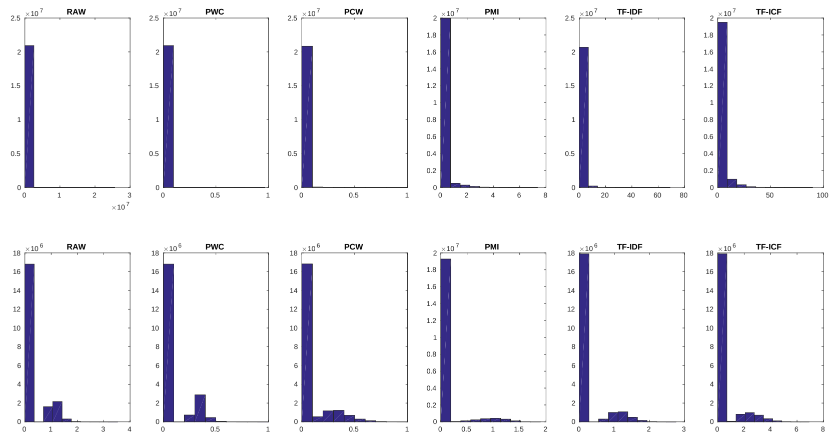


Figure 7. Data distribution (TOP) before and (BOTTOM) after performing power transformation.

word frequencies is mitigated in `RSV` through context selection, transformation, and dimensionality reduction. Among the weighting functions, *tf-icf* results in slightly better results when the number of context units is relatively small. The best result is obtained from word vectors extracted with the *pwc* global weighting function and 50 000 context units. In general, the graph shows that the number of context units is more important than the global weighting function.

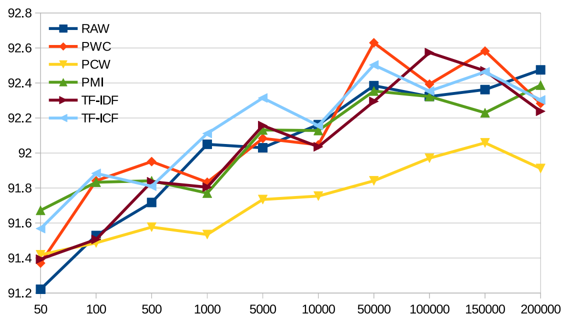


Figure 8. The effect of number of context units (horizontal axis) and global weighting function on the parsing accuracy (vertical axis: UAS). The parsing models are trained evaluated on development set of WSJ using gold part-of-speech tags.

5.3. Context Selection

As described in Sec. 3.1, `RSV` uses an entropy-based context selection that weights each context unit with respect to the entropy of seeing words in its contextual environments and the marginal frequency of the context unit itself (see Eq. 3). In this section, we study the effect of the entropy-based context selection on the word vectors. To this end, we train `RSV` in two ways: one is based on the context units selected by Eq. 3 and the other is based on the top most frequent context units. In order to provide a detailed study of the feature selection methods, we set the number of context units D

to a small value, $D = 1000$, and train RSV on two different corpora, the LDC corpus and the Wikipedia corpus. Both corpora are normalized as described in Sec. 5.1. With this setting, for each corpus, we have two sets of word vectors: 1) the entropy-based vectors (i.e., the vectors trained with the entropy-based feature selection) and 2) the frequency-based vectors (i.e., the vectors trained with the frequency-based feature selection). The word vectors are evaluated with regard to their contribution to the word similarity benchmarks and other tasks such as part-of-speech tagging, named-entity recognition, and dependency parsing described in Sec. 5.1.

Fig. 9 shows the results obtained from the word similarity benchmarks using the word vectors trained with the entropy-based and the frequency-based feature selections. The results show that depending on the training corpus the entropy-based feature selection can have a positive effect on most of the word similarity benchmarks. The positive effect of the entropy-based feature selection is clear when we consider the results obtained from the Wikipedia word vectors. The results obtained from the LDC word vectors are at the same level for both methods. The average of the results obtained from the LDC word vectors is 0.27 for both entropy-based and frequency-based feature selection. This indicates that the entropy-based feature selection has almost no effect on the word vectors extracted from LDC. However, the average of the results obtained from the entropy-based and frequency-based feature selection on the Wikipedia word vectors are 0.20 and 0.18, respectively. This shows that the entropy-based feature selection does not lead to any harm to the word vectors and that it can find a set of informative context units leading to high-quality word vectors.

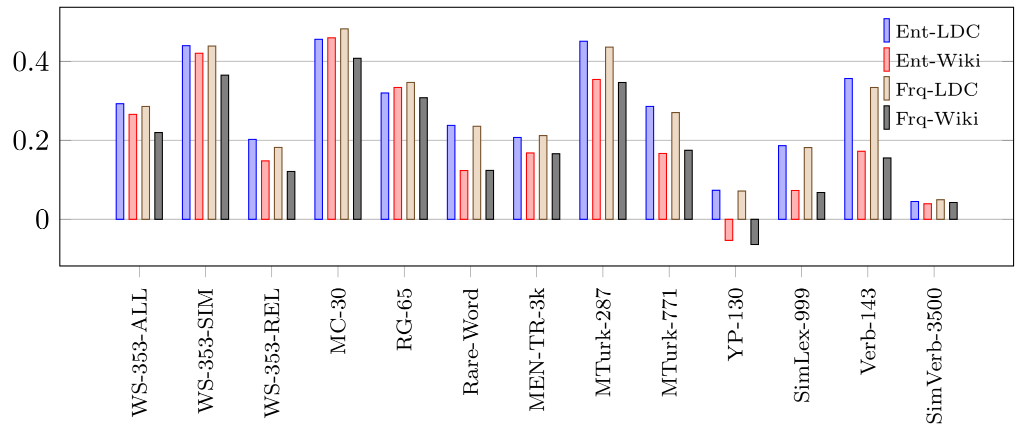


Figure 9. The breakdown results obtained from the word similarity benchmarks with word vectors trained with the *entropy-based feature selection* (Ent) and the *frequency-based feature selection* (Frq). The word vectors are extracted from LDC and Wikipedia corpora. The x-axis shows the word similarity benchmarks and the y-axis shows the correlation between word similarities. Each legend item shows a combination of context selection method (Ent or Frq) and the training corpus (LDC or Wiki).

Table 2 summarizes the results obtained from all sets of word vectors on the word similarity benchmark, part-of-speech tagging, named-entity recognition, and dependency parsing. The results show that the entropy-based feature selection improves the quality of the word vectors when they are assessed on part-of-speech tagging and word similarities. However, we see that it leads to a slight decrease on the results of named-entity recognition and dependency parsing. The reduction in the parsing results is not statistically significant. Hence, we conclude that the entropy-based context selection does not significantly affect the dependency parsing task, but it is slightly harmful to the named-entity recognition task. However, it is beneficial to the part-of-speech tagging task, and depending on the training corpus it can improve the results of the word similarity benchmark.

	WSim	NER	POS	Parsing
Ent-LDC	0.27	95.33	93.37	89.38
Ent-Wiki	0.20	94.97	92.58	89.42
Frq-LDC	0.27	95.59	92.87	89.48
Frq-Wiki	0.18	95.21	91.88	89.44

Table 2. The effect of the entropy-based (Ent) and frequency-based (Frq) feature selections on the word vectors trained on the LDC and the Wikipedia (Wiki) corpora separately. WSim: the average of the word-similarity benchmarks, NER: Named-Entity Recognition, POS: Part-Of-Speech tagging, Parsing: F_1-score of dependency parsing on the development set of our parsing setting with gold part-of-speech tags.

In order to have a better understanding about the context units selected by the entropy-based feature criterion, we compare the set A of context units selected by the entropy-based feature selection and the set B of context units selected by their marginal frequency. In our comparison, we look at the intersection and the difference of two sets. When using LDC as the training corpus, the sets A and B are very similar to each other. The difference between these sets in this case is limited to only two items: $A - B = \{$james, </s>$\}$ and $B - A = \{$rules, independent$\}$. This is the reason why the results obtained from the LDC word vectors are close to each other. It also shows that the frequency of context units plays an important role in the entropy-based context selection. However, among the 1000 top frequent context units selected from the Wikipedia corpus, 974 context units were in both sets (i.e., $\mid A \cap B \mid = 974$). The 26 context units in $A - B$ are $\{$color:decom, format=dmyformat=, flag=rn, l@¸v-, m.p.party, goliyon, qf=, defeatedwon, =ingalls, episodes#red, <num>xlib, students.amy, montazur, </s>, =hollingsworth, sen., rediscoving, thnks, color:laid, color:laun, season#super, color:comm, a-dh-<num>-<num>, vakavulewa, agriculture.pennsylvania, harthac$\}$. Some of these words (e.g., l@¸v-) seem completely meaningless, but the others are meta-data which are used in different regions of Wikipedia documents and they contain information about the words that appear after them. The identification of these sorts of context units are useful for the other tasks such as information retrieval and document classification. The 26 context units in $B - A$ are $\{$stated, bishop, irish, redirect, prize, whose, owned, fall, cover, vol, girl, runs, key, stone, islands, sometimes, williams, contract, gallery, oxford, taking, wales, administration, policy, prime, individual$\}$. This comparison shows that the entropy-based context selection prefers some of the less frequent words that appear as meta-data in the documents to the frequent words. Our experimental results confirm that this selection is beneficial to some tasks such as part-of-speech tagging and word similarity benchmarks.

5.4. Dimensionality

The dimensionality of the word vectors is determined by the number of singular vectors used in Eq. 6. The number of dimensions affects the time required to perform singular value decomposition (SVD). Using the randomized SVD method described by Halko, Martinsson, and Tropp (2011), the extraction of k top singular vectors of an $M \times N$ matrix requires $O(MN \log(k))$ floating point operations. This shows that the cost for having larger word vectors grows logarithmically with the number of dimensions.

The Stanford parser can benefit from the information provided by the high dimensional word vectors if it is trained with enough hidden units to capture the interrelations between the dimensions. Table 3 shows how the parsing accuracy is affected by the dimensionality of word vectors and the number of hidden units. In general, increasing the number of hidden units leads to a more accurate parsing model. However, it is not the same for the number of dimensions. Increasing the dimensionality of word vectors does not necessarily lead to higher parsing accuracy. We see that the best parsing accuracy is obtained from word vectors with only 50 dimensions and 400 hidden units. In order to see if this incremental behaviour continues with more hidden units, we trained the parser with larger numbers of hidden units with maximum 1 500 hidden units. Our empirical results show significant improvement in the parsing accuracy with larger number of hidden units. Using 50-dimensional word vectors and 1, 200 hidden units, we could get the UAS of 93.6 and LAS of 92.2 on the development set of WSJ, annotated with gold part-of-speech tags.

h→	200		300		400	
↓k	UAS	LAS	UAS	LAS	UAS	LAS
50	92.6	91.2	92.8	91.4	**93.1**	**91.8**
100	92.6	91.2	92.8	91.4	93.0	91.6
200	92.5	91.0	92.7	91.4	92.8	91.4

Table 3. The accuracy of parsing models on the development set of WSJ trained with k-dimensional word vectors and h hidden units on the development set of WSJ corpus annotated with gold part-of-speech tags.

These achievements are obtained at the cost of a linear reduction of efficiency in both the training and testing phase of the parser, i.e., the parser becomes slower. 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.

Each pass over the network takes $O(kh+h)$ time. So, each input vector needs $O(3(kh+h)))$ time to be processed by the training algorithm. Given the trained model, the output signals are generated through only one forward pass that takes $O(kh + h)$ time. So, with a fixed number of dimensions, we see only a linear growth in the time required for training and testing the parser which depends on the size of the hidden layer, $O(h)$.

5.5. Results on Test Set

In this section we report the results obtained from the WSJ and UD test sets and compare them with the results reported for other similar parsing architecture. Among the many state-of-the-art parsing architectures, we have chosen Parsito (Straka, Hajic, Straková, & Hajic jr, 2015), UDPipe (Straka et al., 2016), and the Google parsing architecture (Weiss, Alberti, Collins, & Petrov, 2015). The reason for choosing these parsers is that all of them are inspired by the Stanford neural dependency parser (Chen

& Manning, 2014), used in our experiments. Parsito preserves the greedy nature of the Stanford parser and adds two main items to it: the first is a search-based oracle similar to a dynamic oracle and the second is the set of morphological features provided by the corpus of Universal Dependencies (Nivre et al., 2016). It also replaces the cube activation function used in the Stanford parser with the *tanh* function. UDPipe sacrifices the greedy nature of Parsito through adding a beam search decoder to it. The Google parser uses the arc-standard transition system with a deep feed-forward neural network as its classifier. The parser does not use pre-trained word embeddings as input, but instead, it generates the embeddings by another hidden layer, called the *embedding layer.* The parser can be trained in both greedy and non-greedy fashions. In the greedy fashion, in addition to the embedding layer, the parser adds one more hidden layer to the original architecture of the Stanford parser. A perceptron layer is also added to the original architecture which takes the activations of all hidden layers as input and generate a parsing action as output. The perceptron is trained with beam search and early update which makes it a non-greedy parser.

Table 4 compares the results obtained from the Stanford neural dependency parser (Chen & Manning, 2014) trained with our settings and the results reported for the other parsing architectures mentioned above. The superiority of our results to the results reported for original `RSV` (Basirat & Nivre, 2017) shows that the automatic tuning step results in better word vectors. When we compare our results on UD with the other methods, we see the importance of using more effective word vectors together with enough hidden units versus increasing the complexity of parsing architecture.

Corpus	Model	Pred. Tag		Gold Tag	
		UAS	LAS	UAS	LAS
WSJ	Stanford (Chen & Manning, 2014)	91.8	89.6	–	–
	Stanford + RSV (Basirat & Nivre, 2017)	–	–	93.0	91.7
	Stanford + RSV (Our setting)	92.7	90.5	**93.3**	**92.0**
	Google greedy (Weiss et al., 2015)	93.2	91.2	–	–
	Google perceptron (Weiss et al., 2015)	**94.0**	**92.1**	–	–
UD	Parsito standard oracle (Straka et al., 2015)	–	–	86.7	84.2
	Parsito search-based oracle (Straka et al., 2015)	–	–	87.4	84.7
	UDPipe (Straka et al., 2016)	84.0	80.2	87.5	84.9
	Stanford + RSV (Basirat & Nivre, 2017)	84.6	80.9	87.6	84.9
	Stanford + RSV (Our setting)	**85.6**	**81.4**	**88.1**	**85.4**

Table 4. Parsing accuracies on the test set of WSJ and the English UD treebank annotated with both predicted and gold part-of-speech tags.

5.6. Comparison

We compare `RSV` with other methods of word embedding such as:

- `SENNA` (Collobert et al., 2011)
- `GloVe` (Pennington et al., 2014)
- `word2vec` (Mikolov, Chen, et al., 2013), both continuous-bag-of-words (CBOW) and skip-gram (SGRAM)
- `word2vecf` (SGARMF) (Levy & Goldberg, 2014a), an extension of the skip-gram model of `word2vec`
- `HPCA` (Lebret & Collobert, 2014)
- `RI` (Sahlgren, 2006)

Apart from SENNA where we use pre-trained word vectors provided by its authors, all word vectors are generated from the same corpus as that RSV is trained on (see Sec. 5.1). word2vec is trained with symmetric context but the remaining models are trained with asymmetric context. The window size for all models is 1 word. We choose these settings on the basis of the arguments made in Sec. 3.1. The dimensionality of word vectors for all word embedding methods is set to 50 dimensions. In order to provide a comparison between the syntax-based context and the window-based context, we use word2vecf, an extension of the skip-gram model that uses dependency context instead of the window-based context. word2vecf needs a parsed corpus for training. We use the Stanford dependency parser (Chen & Manning, 2014) to parse the raw corpus described in Sec. 5.1 and use the parsed corpus for training word2vecf. We report RSV results obtained from different global weighting functions mentioned in Sec. 5.2. We set the size of the context units D equal to 50 000.

Table 5 gives a comparison between the efficiency and the performance of word embedding methods in different tasks. The columns of the table are divided into two major parts showing the efficiency and the performance of the word embedding methods. The first part shows the time required by each of the word embedding methods to generate a set of word vectors. This part is shown in the column *Time*. The second part shows the performance of word embeddings in the word similarity benchmark and the other tasks described in Sec. 5.1. The rows of the table, listing the word embedding methods, is also divided into two parts. The first part is related to the non-RSV methods and the second is related to different variants of RSV trained with different global weighting function.

Model	Time (Min.)	WSim	POS Tagging	NER	Dep. Parsing (F1-score) Gold	Predicted
SENNA	—	0.4	**96.5**	97.0	91.4	90.4
GLOVE	52	0.4	95.2	**97.4**	91.8	**90.5**
CBOW	145	0.4	96.3	96.2	91.5	90.2
SGRAM	185	**0.5**	96.4	97.2	91.6	90.3
SGRAMF	156	0.4	96.3	97.2	91.4	90.4
HPCA	**45**	0.3	96.3	96.2	90.9	89.9
RI	**46**	0.2	95.4	93.6	91.2	90.0
RAW	52	0.4	96.1	96.5	91.7	**90.5**
PWC	52	0.4	96.1	96.6	**91.9**	90.4
PCW	52	0.3	96.2	96.6	91.0	89.8
PMI	53	0.4	96.2	96.3	91.6	90.4
TF-IDF	53	0.3	96.2	96.6	91.6	90.4
TF-ICF	53	0.3	96.2	96.5	91.8	90.4

Table 5. A comparison between the efficiency and the performance of different word embedding methods. The performance is measured by the contribution of word embeddings on different tasks such as the word similarity (WSim), part-of-speech tagging (POS), named-entity recognition (NER), and dependency parsing with both predicted and gold part-of-speech tags.

The column *Time* shows that HPCA and RI are faster than RSV which is on par with GloVe. As discussed in Sec. 4, all of these methods are closely related to each other. They all need to scan the training corpus once and explicitly build a co-occurrence matrix. Then, they apply a transformation function followed by a dimensionality reduction on the co-occurrence matrix. The reason why HPCA and RI are faster than RSV is that they use a fixed transformation function whose parameters are set by the end user. However, RSV runs a parameter tuning step which searches for the best value of θ in Eq. 4. If the efficiency of the word embedding is of interest, one can disable the parameter tuning step and use RSV with the default value of $\theta \approx 0.073$. In this case, the time required by RSV is comparable with RI and HPCA.

The results showing the performance of the word vectors confirm the observation made by Schnabel, Labutov, Mimno, and Joachims (2015): "*different tasks favor different embeddings*". The best results for the word similarity benchmark are obtained from word2vec with Skip-gram architecture. The results obtained from RSV are as good as the other methods. SENNA results in the highest accuracy on the part-of-speech tagging task. In comparison with SENNA, RSV shows relatively weaker performance on the part-of-speech tagging but in general the results obtained from RSV are comparable or higher than the results obtained from the other methods. We see a large variation in the results obtained from the named-entity recognition task. The best result on this task is obtained from GloVe which is distinctively higher than other methods such as RSV. This is different from the results obtained from part-of-speech tagging, where GloVe shows very weak performance on it but RSV is comparably good. The performance of RSV on the dependency parsing task is as good as other methods such as GloVe. We see that word2vecf (SGRAMF) does not necessarily result in higher performance on the dependency parsing task although it uses the dependency context.

To decide whether the slightly higher results of RSV on the dependency parsing task are due to chance or not, we perform a bootstrap statistical significance test (Berg-Kirkpatrick, Burkett, & Klein, 2012) on the results. The null hypothesis of the test is H_0: *RSV is not better than B*, where B can be any of the aforementioned word embeddings methods. The results show that the null hypothesis is rejected for SENNA, HPCA, and RI with quite high confidence. However, the null hypothesis is not rejected with the standard significance level $\alpha = 0.05$ for other methods of word embedding. This shows that the contribution of RSV to the task of dependency parsing is as good or better than other methods of word embeddings. Table 6 shows the p-values.

	SENNA	GLOVE	CBOW	SGRAM	SGRAMF	HPCA	RI
p	0.00	0.40	0.20	0.28	0.18	0.00	0.00

Table 6. p-value of the null hypothesis H_0: *RSV is not better than the other word embedding methods.*

6. Conclusion

In this paper, we have proposed a word embedding method that extracts a set of word vectors from a distributional semantic space. The context units (words) corresponding to the dimensions of the semantic space are selected not only on the basis of their frequencies but also on the basis of the amount of certainty involved in their contextual environments. It has been empirically shown that the number of context units is the only tuning parameter of the word embedding method. Using these context units, a set of initial high-dimensional word vectors are formed. The elements of these vectors are a function of the frequency of seeing words in the domain of the context units. The distribution of these vectors is highly skewed toward the null vector of the semantic space. This problem is mitigated by a power transformation function whose degree is automatically determined by simulated annealing. The final low dimensional word vectors are formed with right singular vectors of the matrix formed by the transformed vectors. The word vectors have been evaluated with regard to their contributions to the dependency parsing task. The experimental results show that the word vectors are highly influenced by the proposed context selection and the transformation steps. The word embedding method has also been compared with other popular word embedding methods. The comparison was made with regard to the efficiency and the performance of word embedding methods on different downstream tasks. In comparison with other word embedding methods, it has been shown that the proposed method is on a par with other word embedding methods.

References

Achlioptas, D. (2001). Database-friendly random projections. In *Proceedings of the 20th acm sigmod-sigact-sigart symposium on principles of database systems* (pp. 274–281).

Basirat, A., & Nivre, J. (2017). Real-valued syntactic word vectors (RSV) for greedy neural dependency parsing. In *Proceedings of the 21th nordic conference on computational linguistics (NoDaLiDa)* (pp. 20–28).

Berg-Kirkpatrick, T., Burkett, D., & Klein, D. (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* (pp. 995–1005).

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*, 993–1022.

Chen, D., & Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 740–750).

Church, K. W., & Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational linguistics*, *16*(1), 22–29.

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

de Marneffe, M.-C., MacCartney, B., & Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of the fifth international conference on language resources and evaluation (LREC 2006)* (pp. 449–454). Genoa, Italy: European Language Resources Association (ELRA).

de Marneffe, M.-C., & Manning, C. D. (2008). The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on cross-framework and cross-domain parser evaluation* (pp. 1–8).

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., & Smith, N. A. (2015). Transition-based dependeny parsing with stack long short-term memory. In *Proceedings of the 53rd annual meeting of the association for computational linguistics (ACL) and the 7th international joint conference on natural language processing* (pp. 334–343).

Faruqui, M., & Dyer, C. (2014). Community evaluation and exchange of word vectors at wordvectors.org. In *Proceedings of the 52nd annual meeting of the association for computational linguistics: System demonstrations* (p. 19-24).

Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, *53*(2), 217-288.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hofmann, T. (1999). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international acm sigir conference on research and development in information retrieval* (pp. 50–57).

Jolliffe, I. (2002). *Principal component analysis.* Springer-Verlag.

Kiela, D., & Clark, S. (2014). A systematic study of semantic vector space model parameters. In *Proceedings of the 2nd workshop on continuous vector space models and their compositionality (cvsc) at conference of the european chapter of the association for computational linguistics (eacl)* (pp. 21–30).

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

Lebret, R., & Collobert, R. (2014). Word embeddings through Hellinger PCA. In *Proceedings of the 14th conference of the european chapter of the association for computational linguistics (EACL)* (pp. 482–490).

Lebret, R., & Collobert, R. (2015). Rehabilitation of count-based models for word vector representations. In *Computational linguistics and intelligent text processing* (pp. 417–429).

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9–50). Springer Berlin Heidelberg.

Levy, O., & Goldberg, Y. (2014a). Dependency-based word embeddings. In *Proceedings of the 52nd annual meeting of the association for computational linguistics* (pp. 302–308).

Levy, O., & Goldberg, Y. (2014b). Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems* (pp. 2177–2185).

Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, *28*(2), 203–208.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993, June). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics - Special issue on using large corpora*, *19*(2), 313 - 330.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of workshop at the international conference on learning representations (ICLR).*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems 26* (pp. 3111–3119). Curran Associates, Inc.

Nayak, N., Angeli, G., & Manning, C. D. (2016). Evaluating word embeddings using a representative suite of practical tasks. In *Proceedings of the 1st workshop on evaluating vector-space representations for nlp, august 2016* (pp. 19–23).

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the workshop on incremental parsing: Bringing engineering and cognition together* (pp. 50–57).

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., ... others (2016). Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th international conference on language resources and evaluation (lrec 2016)* (p. 1659-1666).

Padó, S., & Lapata, M. (2007). Dependency-based construction of semantic space models. *Computational Linguistics*, *33*, 161–199.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543).

Reed, J. W., Jiao, Y., Potok, T. E., Klump, B. A., Elmore, M. T., & Hurson, A. R. (2006). Tf-icf: A new term weighting scheme for clustering dynamic data streams. In *2006 5th international conference on machine learning and applications (icmla'06)* (p. 258-263).

Roweis, S. T., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500), 2323–2326.

Sahlgren, M. (2006). *The word-space model* (Unpublished doctoral dissertation). Stockholm University.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, *24*(5), 513–523.

Schnabel, T., Labutov, I., Mimno, D., & Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in* *natural language processing* (pp. 298–307).

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5), 1299–1319.

Schütze, H. (1992). Dimensions of meaning. In *Proceedings of the 1992 ACM/IEEE conference on supercomputing* (pp. 787–796).

Straka, M., Hajic, J., Straková, J., & Hajic jr, J. (2015). Parsing universal dependency treebanks using neural networks and search-based oracle. In *International workshop on treebanks and linguistic theories (tlt14)* (p. 208-220).

Straka, M., Hajic, J., & Straková, J. (2016). Udpipe: Trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the tenth international conference on language resources and evaluation (LREC 2016).*

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on natural language learning at hlt-naacl 2003 - volume 4* (pp. 142–147).

Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology-volume 1* (pp. 173–180).

Weiss, D., Alberti, C., Collins, M., & Petrov, S. (2015). Structured training for neural network transition-based parsing. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 323–333).