Sketching Word Vectors Through Hashing

Behrang QasemiZadeh  
DFG SFB 991  
Universität Düsseldorf  
Düsseldorf, Germany  
zadeh@phil.hhu.de

Laura Kallmeyer  
DFG SFB 991  
Universität Düsseldorf  
Düsseldorf, Germany  
kallmeyer@phil.hhu.de

Peyman Passban  
ADAP Centre  
Dublin City University  
Dublin, Ireland  
ppassban@computing.dcu.ie

Abstract

We propose a new fast word embedding technique using hash functions. The method is a derandomization of a new type of random projections: By disregarding the classic constraint used in designing random projections (i.e., preserving pairwise distances in a particular normed space), our solution exploits extremely sparse non-negative random projections. Our experiments show that the proposed method can achieve competitive results, comparable to neural embedding learning techniques, however, with only a fraction of the computational complexity of these methods. While the proposed derandomization enhances the computational and space complexity of our method, the possibility of applying weighting methods such as positive pointwise mutual information (PPMI) to our models after their construction (and at a reduced dimensionality) imparts a high discriminatory power to the resulting embeddings. Obviously, this method comes with other known benefits of random projection-based techniques such as ease of update.

1 Introduction

Word embedding techniques (i.e., using distributional frequencies to produce word vectors of reduced dimensionality) have become a cornerstone of modern natural language processing and information retrieval systems. These quantifications of words are often justified by Harris’ Distributional Hypothesis [Harris, 1954]. It is hypothesized that words of comparable linguistic properties appear with/within a similar set of ‘contexts’. For instance, words of similar meaning co-occur with a similar set of context words \( \{c_1, \ldots, c_n\} \). This hypothesis implies that if this set of context words are partitioned randomly into \( m \) buckets, e.g. \( \{\{c_1, \ldots, c_x\}, \ldots, \{c_y, \ldots, c_n\}\}\), then co-related words still co-occur with similar set of buckets. We design a new random projection to exploit this presumption, and accordingly, propose a new hash-based word embedding technique.

In the remaining of this paper, we propose the method in Section 2. Section 3 provides a brief overview of related work. Section 4 reports results from a number of experiments, followed by a conclusion in Section 5.

2 The Proposed Embedding Technique

Let’s assume that we build an \( m \)-dimensional embedding for a word \( w \) that is co-occurred with a number of context words \( c \). To build an embedding for each \( w \) (denoted by \( \vec{w} \)), the following steps are taken:

1. Initialize a zero \( m \)-dimensional vector \( \vec{w} \)
2. for each \( c \) co-occurred with \( w \) do
3. \( d \leftarrow \text{abs} (\text{hash}(c) \% m) \)
4. \( \vec{w}_d = \vec{w}_d + 1 \)
5. return \( \vec{w} \)

Above, \( w_d \) is the \( d \)th component of \( \vec{w} \); the \text{hash} function assigns a (ideally unique, and independently and identically distributed) hash code (e.g., an integer) to each context word (object). The \text{abs} function returns the absolute value of an input number; \( \% \) is the modulus operator and it gives the remainder of the division of the generated hash code by the chosen value \( m \). We choose Jenkins hash function in our implementation since it shows a low collision rate for short words (bytes sequences):\(^1\)

\[
\text{int} \hash (\text{byte[]} \text{key}) \{ \\
\ \ \ \ \text{int} \ i = 0; \\
\ \ \ \ \text{int} \ \text{hash} = 0; \\
\ \ \ \ \text{while} \ (i != \text{key}.\text{length}) \{ \\
\ \ \ \ \ \ \text{hash} += \text{key}[i++]; \\
\ \ \ \ \ \ \text{hash} += \text{hash} << 10; \\
\ \ \ \ \ \ \text{hash} ^= \text{hash} >> 6; \\
\ \ \ \ } \\
\ \ \ \ \text{hash} += \text{hash} << 3; \\
\ \ \ \ \text{hash} ^= \text{hash} >> 11; \\
\ \ \ \ \text{hash} += \text{hash} << 15; \\
\ \ \ \ \text{return} \text{hash}; \\
\}
\]

Once these embeddings are constructed, similarities between them are given by an appropriate correlation measure such as Goodman and Kruskal’s \( \gamma \) coefficient [Goodman and Kruskal, 1954] (or, alternatively, Kenall’s \( \tau_b \) [Kendall, 1938]). This choice is motivated by the fact that the resulting embeddings are multinomial categorical variables

\(^1\)Designed by Bob Jenkins (2006); see http://www.burtleburtle.net/bob/hash/doobs.html
(of a discrete distribution) with non-Gaussian long-tailed distribution (which are, in many cases, sparse). To compute \( \gamma \), **concordant** and **discordant** pairs must be counted. Given any pairs such as \( (x_i, y_i) \) and \( (x_j, y_j) \) from two \( m \)-dimensional vectors \( \vec{x} \) and \( \vec{y} \) and the value \( v = (x_i - x_j)(y_i - y_j) \), these two pairs are concordant if \( v > 0 \) and discordant if \( v < 0 \). If \( v = 0 \), the pair is neither concordant nor discordant. Let \( p \) and \( q \) be the number of concordant and discordant pairs, then \( \gamma \) is given by [Chen and Popovich, 2002, p. 86]:

\[
\gamma = \frac{p - q}{p + q}.
\]

As with other statistical models, normalizing the counted frequencies in the obtained embeddings using a ‘weighting process’ enhances results. Evidently, a weighting process can eliminate uninformative and irrelevant frequencies and boost the impact of informative and discriminatory ones, e.g., through normalizing raw frequencies by the expected and marginal frequencies, such as the popular PPMI weighting [Turney, 2001] [Bouma, 2009]. If \( W_{p \times m} \) is the matrix consisting of \( p \) row vectors \( \vec{w} \) (i.e., the set of obtained embeddings in our model), the PPMI weight for a component \( w_{xy} \) in \( W \) is:

\[
ppmi(w_{xy}) = \max(0, \log \frac{w_{xy} \times \sum_{i=1}^{p} \sum_{j=1}^{m} w_{ij}}{\sum_{i=1}^{p} w_{yi} \times \sum_{j=1}^{m} w_{xj}}).
\]

For the weighted vectors (which often have rectified Gaussian distribution), we expect a correlation measure such as Pearson to map the high-dimensional \( \vec{w} \) onto a space of reduced dimensionality \( W \). Random projections and hash kernels have been vibrant research areas in artificial intelligence and theoretical computer science. These methods have been employed to provide viable solutions for a range of problems requiring a notion of (approximate) nearest neighbor search, particularly in information retrieval tasks, e.g., identification of duplicate and near duplicate documents [Manku et al., 2007] and string matching [Dalvi et al., 2013] [Michael, 1981], semantic labeling [Yang et al., 2016], and cross-media retrieval [Wang et al., 2015] to name a few. Naturally, these methods have also been applied to the problem of word embedding, either as a dimensionality reduction method [Bingham and Mannila, 2001] [Kaski, 1998], or in the form of an incremental vector space construction technique in which the random projection-based dimensionality reduction process is merged with the process of collecting word co-occurrence frequencies, e.g., [Kanerva et al., 2000] [Geva and De Vries, 2011] [Q. Zadeh and Handschuh, 2014].

**2.1 Method’s Justification**

The proposed method can be explained mathematically using the principles of dimensionality reduction using random projections.

Let \( C_{p \times n} \) denotes the set of \( p \) word vectors obtained by counting the co-occurrences of each target word \( \vec{w}_i \) (1 \( \leq i \leq p \)) with each context element \( c_j \) (1 \( \leq j \leq n \)). In most applications, \( n \) is a very large number in a way that it hinders processes (and causes the so-called curse of dimensionality problem). To address this problem, \( C_{p \times n} \) undergoes a set of ‘transformations’ \( T \) to map the high-dimensional space \( C_{p \times n} \) onto a space of reduced dimensionality \( W_{p \times m} \) (\( m \ll n \)): \( C_{p \times n} \times T_{n \times m} = W_{p \times m} \). In our proposed method, \( T_{n \times m} \) is a sparse randomly generated matrix, in which \( t_{ij} \) elements of \( T \) has the following distribution:

\[
t_{ij} = \begin{cases} 
0 & \text{with probability } \frac{m-1}{m}, \\
1 & \text{with probability } \frac{1}{m},
\end{cases}
\]

such that each row vector of \( T \) has exactly one component that has value 1, and these non-zero values of \( T \) have independent and identical distribution. It can be verified that the procedure proposed in the previous section computes the desired embeddings (i.e., \( W \)) by (a) de-randomization of \( T \) using a hash function and the modulus operator, and (b) serializing the involved multiplication for computing \( C \times T \) to a set of addition operations (based on the distributive property of multiplication over addition).

As an expert can identify, the major novelty of our method is in the way that we compute \( T \). Previously, random-projection-based methods compute \( T \) with the goal of having the least distortions in pairwise (\( \alpha \)-normed) distances while mapping from \( C \) onto \( W \) — e.g., the well-known lemmas proposed in [Johnson and Lindenstrauss, 1984] (for \( \ell_2 \)-normed spaces), [Indyk, 2000] (for \( \ell_1 \)-normed spaces), and their subsequent refinements and generalizations such as [Li et al., 2006]. In contrast to these research, we disregard this classic desiderata of preserving distances and the goal of minimum-distortion correspondence. In return, we motivate our proposed random projection directly using the implications that Harris’ Distributional Hypothesis bears.

**3 Related Work**

Random projections and hash kernels have been vibrant research areas in artificial intelligence and theoretical computer science. These methods have been employed to provide viable solutions for a range of problems requiring a notion of (approximate) nearest neighbor search, particularly in information retrieval tasks, e.g., identification of duplicate and near duplicate documents [Manku et al., 2007] and string matching [Dalvi et al., 2013] [Michael, 1981], semantic labeling [Yang et al., 2016], and cross-media retrieval [Wang et al., 2015] to name a few. Naturally, these methods have also been applied to the problem of word embedding, either as a dimensionality reduction method [Bingham and Mannila, 2001] [Kaski, 1998], or in the form of an incremental vector space construction technique in which the random projection-based dimensionality reduction process is merged with the process of collecting word co-occurrence frequencies, e.g., [Kanerva et al., 2000] [Geva and De Vries, 2011] [Q. Zadeh and Handschuh, 2014]. [Kanerva et al.] employed sparse Gaussian random projections [QasemiZadeh, 2015] to build word vectors directly at a reduced dimensionality and showed that their proposed ‘random indexing’ technique yields results comparable to the latent semantic analysis technique of [Deerwester et al., 1990], which employs truncation using singular value decomposition for dimensionality reduction. By the same token, [Geva and De Vries] and [Q. Zadeh and Handschuh] extend the idea proposed by [Kanerva et al.] for the comparison of textual similarities using the Hamming and Manhattan distances, respectively. However, when it comes to comparing semantic similarities between words, these methods fail to compete with the more recent neural-based embedding techniques, e.g., word2vec [Mikolov et al., 2013b] and GloVe [Pennington et al., 2014]. It is known that crude distances between words are not discriminatory enough to address the
so-called semantic relatedness tasks, and a weighting process such as PPMI is crucial for achieving a high score. Unfortunately, since these methods use projections with zero expectation, the sum of components in their resulting vectors is always zero. Thus, weighting techniques such as PPMI cannot be applied to these models after their construction (simply, due to the problem of division by zero). Compared to these techniques, the method proposed in this paper has a better computational complexity (thanks to its sparser projections) and yields better results in semantic relatedness tasks given the possibility of applying weighting techniques after the construction of randomly projected spaces.

4 Empirical Evaluation

We report results from empirical evaluations in both intrinsic and extrinsic setups. Besides an intrinsic evaluation using word relatedness tests, we report results when the method is used for training a neural-based part-of-speech tagger.

4.1 Intrinsic Evaluation

Datasets and Material: We evaluate our method over a number of word relatedness tests. Each relatedness test encompasses a set of word pairs, associated with similarity/relatedness ratings obtained from human annotators. For each test, evaluation takes the form of calculating the harmonic mean of Pearson and Spearman’s correlations between the list sorted by the human-induced scores and the list sorted by the scores assigned to word pairs by the system. Relatedness tests in our experiments are WS353 [Finkelstein et al., 2001], WSS and WSR by [Agirre et al., 2009], the classic tests of MC30 [Miller and Charles, 1991] and RG65 [Rubenstein and Goodenough, 1965], the Stanford Rare Word dataset RW [Luong et al., 2013], M287 by [Radinsky et al., 2011], M771 [Halawi et al., 2012], SE [Hill et al., 2015], YP130 verb relatedness [Yang and Powers, 2006], and MEN [Bruni et al., 2014]. As input corpus, we build embeddings from a recent dump of Wikipeda (January 1st, 2017; 2.9 billion tokens).

Baselines: For this input, as one baseline, we train a word2vec CBOW model [Mikolov et al., 2013b], one of the most popular word embedding techniques. To train this model, we use parameters that are commonly known to give best performances across the chosen tasks in our experiments (i.e., 10 negative samples, 1e-5 for subsampling, and context-windows that extend 7 tokens around target words). We set the dimensionality of this model to 500 (put aside its impact on computational complexity, for our input corpus and the targeted tests, we observed that using dimensionality larger than 500 has an adverse effect on word2vec’s CBOW algorithm and deteriorates its performance). We use W2C-CBOW to denote the obtained set of embeddings from this trial.

As an additional baseline, we report the performance of a classic unweighted and PPMI-weighted high-dimensional model trained over the same input corpus. In this case, the dimensionality of the classic model soared to 10.671 million.

Result: We build word vectors of dimension $m = 500$, 1000, and 2000, and 4000 using our proposed hash-based embedding technique. Similar to the classic baseline method, except a white space tokenization, we do not use any additional heuristics or information (such as subsampling criteria used in word2vec). Table I summarizes the observed results, as well as the average of performances across these tasks.

As shown, disregarding $m$ and on average, our unweighted and PPMI-weighted models outperform their counterpart classic high-dimensional models. Moreover, for large values of $m$, weighted models tend to perform as well as word2vec’s CBOW model. Interestingly, increasing $m$ hurts the task-based performances for unweighted models, while it has a positive impact on the PPMI-weighted models. Overall, when it comes to choosing a neural embedding learning method versus our method for addressing a semantic similarity task, our experiments suggest that the hash-based method is more suitable for similarity identification than relatedness (in the sense that described in [Agirre et al., 2009]), and for tasks that involve comparison of frequent and infrequent words (due to the nature of PPMI weighting). Naturally, the hash-based method is a better choice than a neural embedding learning technique when dealing with frequently updated text data and streams 3 and when computational resources for training neural nets are insufficient or the learning process is constrained by hard time limits (note that hash-based embeddings can be fetched at any time during the construction of vectors).

In our experiments and using the specific configurations, building word vectors using the hash-based method took almost one-third of the time for building the CBOW model (using 16 threads on a dual CPU machine). Note that the computational complexity of our method, in contrast to word2vec, is independent of $m$. For $m = 500$, our method requires half of the memory used by word2vec using a sparse vector representation model (note that we use unsigned integers for collecting co-occurrence frequencies and postpone using float number arithmetic until we weight vectors); this is not surprising given the very long tail of co-occurrence frequencies. Word2vec’s CBOW-induced vectors (like other neural embeddings) are dense: in contrary to our method, even for words of low frequencies, all the elements of neural-based embeddings are non-zero. We would like to emphasize that

---

3Note that for these methods, projections with zero expectation is essential for achieving an acceptable performance when computing projected spaces (i.e., to guarantee the sparsity of projection matrices).

3And, in scenarios in which vectors are built using a strategy other than offline scanning large text corpora, e.g., as explained in [Turney, 2001] by querying a Web search engine.
the real computational advantages of the proposed method become apparent when building vectors for real large corpora and vocabularies. At any rate, a neural embedding method such as word2vec requires two passes over an input corpus: one pass to make a frequency profile of words (which is used, e.g., for subsampling), and the second pass to adjust and optimise weights (i.e., the actual training phase). This requirement, however, is not necessary for our method (although a two-pass strategy for subsampling and the selection of a subset of co-occurrence events can be employed to enhance discriminatory power of the hash-based embeddings).

Regarding the classic method, compared to the hash-based method, building vectors and frequency profiles may require slightly less amount of processes (given that, theoretically[^1], no hash code must be computed); however, compared to our method, storing these vectors demands much more amount of memory. Moreover, the process of weighting these large vectors is computationally intense and time-consuming. In comparison, using the proposed hash-based method dramatically reduces the time required for the weighting process. For instance, in our experiments, PPMI weighting of high-dimensional vectors took almost two hours while this process was carried out in only a few minutes using the hash-based method.

### Boosting Performances in Relatedness Tasks

Through several means, the performance of the proposed hash-based embedding method for semantic similarity tasks can be improved. The easiest method is, perhaps, using models of higher dimensionality (as results reported in the previous section imply), or/and to enlarge the size of input corpora. As reported in Table 1, we observe that increasing the dimensionality together with PPMI weighting of models enhance task-based performances. Obviously, increasing the dimensionality does not affect the computational complexity of building a model; however, it increases the cost of similarity computations.

[^1]: We say ‘theoretically’ since an inverted index of context elements (often using hashing) is necessary for building classic models.

### Table 1: Evaluation Results

| Vector Set | YP130 | RW | M287 | M771 | MC30 | MEN | WS353 | WSS | WSR | RG65 | SL | Mean | A | G |
|------------|-------|----|------|------|------|-----|-------|-----|-----|------|----|-----|---|---|
| Baseline   |       |    |      |      |      |     |       |     |     |      |    |     |   |   |
| Classic-Unweighted | 0.21 | 0.07 | 0.24 | 0.16 | 0.28 | 0.22 | 0.30  | 0.39 | 0.21 | 0.28 | 0.11 | 0.224 | 0.204 |
| Classic-PPMI   | 0.47 | 0.32 | 0.57 | 0.57 | 0.75 | 0.68 | 0.49  | 0.53 | 0.63 | 0.76 | 0.25 | 0.547 | 0.521 |
| W2V-CBOW     | 0.45 | 0.22 | 0.53 | 0.64 | 0.72 | 0.76 | 0.70  | 0.78 | 0.65 | 0.75 | 0.40 | 0.600 | 0.567 |
| Our-Method (Unweighted) | | | | | | | | | | | | | |
| Dimension=500 | 0.32 | 0.14 | 0.36 | 0.44 | 0.38 | 0.53 | 0.40  | 0.48 | 0.36 | 0.70 | 0.28 | 0.417 | 0.387 |
| Dimension=1000 | 0.34 | 0.12 | 0.37 | 0.40 | 0.63 | 0.50 | 0.40  | 0.48 | 0.36 | 0.65 | 0.24 | 0.408 | 0.375 |
| Dimension=2000 | 0.32 | 0.12 | 0.41 | 0.40 | 0.59 | 0.48 | 0.37  | 0.44 | 0.36 | 0.60 | 0.22 | 0.392 | 0.361 |
| Dimension=4000 | 0.33 | 0.15 | 0.44 | 0.38 | 0.59 | 0.46 | 0.33  | 0.38 | 0.34 | 0.62 | 0.19 | 0.383 | 0.356 |
| PPMI (Unweighted) | | | | | | | | | | | | | |
| Dimension=500 | 0.39 | 0.28 | 0.61 | 0.61 | 0.64 | 0.71 | 0.63  | 0.67 | 0.65 | 0.75 | 0.33 | 0.570 | 0.545 |
| Dimension=1000 | 0.45 | 0.29 | 0.58 | 0.55 | 0.73 | 0.72 | 0.64  | 0.71 | 0.64 | 0.75 | 0.34 | 0.582 | 0.558 |
| Dimension=2000 | 0.47 | 0.30 | 0.62 | 0.57 | 0.75 | 0.73 | 0.65  | 0.72 | 0.65 | 0.76 | 0.35 | 0.597 | 0.573 |
| Dimension=4000 | 0.49 | 0.30 | 0.61 | 0.57 | 0.75 | 0.74 | 0.69  | 0.75 | 0.68 | 0.79 | 0.34 | 0.610 | 0.584 |

[^1]: We say ‘theoretically’ since an inverted index of context elements (often using hashing) is necessary for building classic models.

---

Figure 1: Changes of the averaged performances across tasks when increasing dimensionality of models (1a), and increasing the size of input corpus (1b). In Figure 1b, the black and red lines plot averaged performances for models of dimensionality $m = 500$ and $2000$, respectively.
computation. Figure 1a plots changes in the method’s performance (i.e., the arithmetic average of performances across tasks in our experiments) when the dimensionality of models are increased (the evaluation setup, i.e., the input corpus and context window size, remains the same as those reported earlier). As shown, up to \( m = 13000 \), the performance mostly improves. We admit that this solution may not look desirable at first sight. However, we notice that these models of higher dimensionality (e.g., \( m = 4000 \)) can be compressed (e.g., to \( m = 250 \)) using matrix factorization techniques without hurting task-based performances. Hence, we suggest that the proposed hash-based solution can safely replace methods of constructing classic PPMI-weighted high-dimensional models, for which the obtained models are compressed using matrix factorization methods. To show the effect of enlarging the size of input corpus on the method’s performance, in addition to the Wikipedia corpus that we used initially, we feed another 4 billion tokens of web crawled text data to our models. As shown in Figure 1b, disregarding the dimension of models (here 500, and 2000), using larger input corpora enhances the method’s performance in relatedness tasks.

Second, like many embedding techniques, our method can be tuned for each task, e.g., by choosing an appropriate context window size, and in general, by making (let’s call it) linguistically more informed decisions. For instance, we could boost the performance in the MEN relatedness from 0.71 to 0.76 using larger context window (i.e., 13+11 instead of 5+5). Similarly, we could enhance the obtained performance for the YP130 verb relatedness from 0.39 to 0.72 (for \( m = 500 \)) through filtering context words using dependency parses (context elements were limited to those words with direct syntactic relationships to the targeted verbs). Apart from these, our method allows for heterogeneous context types: For example, in addition to context words, we use document-level co-occurrences to boost performance in the WS353 test from 0.67 to 0.72. For each \( w \) appeared in document \( d_i \), we pass to the hash function \( d_i \)’s identifier in our input and continue updating \( \bar{w} \) as instructed in the algorithm. Note that there is no limit on the type of context elements that can be fed to our method as long as these elements can be converted to byte sequences and subsequently to a hash code. In comparison, this is not that straightforward with neural embedding techniques: often the objective function of neural nets (which defines their underlying optimization goal) must be modified and adapted for new context types.

Apart from methods discussed above, ‘retrofitting’ (refining word embeddings using lexical relations available in lexical resources [Faruqui et al., 2015]) is another methodology for improving results. Our method can be easily adapted for retrofitting. We implement a notion of retrofitting by sim-ply treating lexical resources like any other text files; however, instead of scanning these resources using a sliding context window, we make sure that we encode co-occurrence information about all related items in each entry of an input lexical resource. For example, given a synset \( S \) of words in WordNet (i.e, \( S = \{ w_1 \ldots w_n \} \)), for each \( w_i \in S \), we consider all the remaining words in \( S \) as context element and respectively update \( \bar{w}_i \) with these context elements. To show the impact of this ‘retrofitting’ method, we take vectors of dimensionality \( m = 500 \) from our earlier experiment, and update them by reading WordNet [Miller, 1995] and PPDB [Ganitkevitch et al., 2013] (i.e., in addition to the Wikipedia corpus, WordNet and PPDB are also fed to our algorithm). Table 2 reports our results. As discussed in [Faruqui et al., 2015], encoding this knowledge into neural embedding models is not straightforward, and it demands certain modification in their learning algorithm. For instance, we observe that retraining the CBOW algorithm with WordNet and PPDB as additional inputs leads to a negligible improvement on task-based performances (an improvement of 0.002 on average), whereas performance gain in retrofitted hash-based model is considerably large (from 0.417 to 0.436 for the unweighted model, and from 0.570 to 0.647 for the PPMI-weighted model). In this experiment, we provide the word2vec’s CBOW algorithm with the same input used for building our hash-based model (i.e., in addition to the Wikipedia corpus, we list the set of all generated tuples from the WordNet and PPDB as input).

4.2 Extrinsic Evaluation

When developing models for upstream NLP tasks, an important application of embedding learning methods is to provide a numerical representation of words for employed (neural) learning algorithms. Likewise relatedness tasks, the word2vec family of embeddings methods [Mikolov et al., 2013a; Le and Mikolov, 2014] are a popular choice for this pre-training phase. Hence, we provide a comparison between the embeddings obtained using the proposed method in this paper and those of the word2vec model in a part-of-speech tagging task. We witness that vectors produced using our hash-based technique are more suitable than word2vec embeddings, at least, in our part-of-speech tagging experiment using Long Short Term Memory (LSTM) units [Hochreiter and Schmidhuber, 1997].

The configuration used for obtaining the embeddings is similar to one explained in the previous section, and the hash-based embeddings are weighted using PPMI prior to feeding them to the network. However, to make sure that the evaluation setup is realistic, we set the size of embeddings to 300 for both of the hash-based and word2vec models. For evaluation, we use the tagged version of the Brown corpus [Francis and Kucera, 1979], which has 57,067 sentences and 313 different tags. Note that although the tag set consists of only 85 categories, the tagged Brown corpus contains combined tags for contracted words; e.g., the sequence of we’ll is tagged as PFSS+MD (which means, personal pronoun followed by modal auxiliary); hence, the number of tags in our experiment increases to 313. We randomly select 2000 sentences as the validation set, another 2000 as the test set, and the remaining sentences are used for training.

The neural tagger has a straightforward architecture and consists of four layers, of which the first layer is a look-up table. This look-up table is a trainable weight matrix whose values are updated during training. Embeddings reside in this look-up table, which are retrieved at each forward pass. The second and third layers include LSTM units of size 300. The last Softmax layer scales values of the network to the range of
A method for sketching word vectors using a hash-based algorithm is proposed. The proposed method is fast, and it requires a small amount of computational resources to build a model; yet, as shown empirically, it can deliver an acceptable performance in both intrinsic and extrinsic evaluation setups. Thanks to the random projection that this method implements, developing embeddings requires no offline training time: vectors can be used, updated, added, and removed at any stage during a system’s life cycle. The method can be particularly useful when building embeddings from text streams or very very large corpora, or when the training of embeddings is constrained by time.

Table 2: Results from retrofitting: Both word2vec-CBOW and hash-based models of dimensionality 500 are trained on the Wikipedia corpus as well as WordNet and PPDB. Comparing these results to those reported in Table 1 shows a considerable improvement of the hash-based method’s performance.

Table 3: Results from the POS-tagging experiment. Numbers in the table indicate the accuracy score.

References
