Improving Distributional Similarity with Lessons Learned from Word Embeddings

Omer Levy **Yoav Goldberg** **Ido Dagan**
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
{omerlevy,yogo,dagan}@cs.biu.ac.il

Abstract

Recent trends suggest that neural-network-inspired word embedding models outperform traditional count-based distributional models on word similarity and analogy detection tasks. We reveal that much of the performance gains of word embeddings are due to certain system design choices and hyperparameter optimizations, rather than the embedding algorithms themselves. Furthermore, we show that these modifications can be transferred to traditional distributional models, yielding similar gains. In contrast to prior reports, we observe mostly local or insignificant performance differences between the methods, with no global advantage to any single approach over the others.

1 Introduction

Understanding the meaning of a word is at the heart of natural language processing (NLP). While a deep, human-like, understanding remains elusive, many methods have been successful in recovering certain aspects of similarity between words.

Recently, neural-network based approaches in which words are "embedded" into a low-dimensional space were proposed by various authors (Bengio et al., 2003; Collobert and Weston, 2008). These models represent each word as a d-dimensional vector of real numbers, and vectors that are close to each other are shown to be semantically related. In particular, a sequence of papers by Mikolov et al. (2013a; 2013b) culminated in the skip-gram with negative-sampling training method (SGNS): an efficient embedding algorithm that provides state-of-the-art results on various linguistic tasks. It was popularized via `word2vec`, a program for creating word embeddings.

A recent study by Baroni et al. (2014) conducts a set of systematic experiments comparing `word2vec` embeddings to the more traditional distributional methods, such as pointwise mutual information (PMI) matrices (see Turney and Pantel (2010) and Baroni and Lenci (2010) for comprehensive surveys). These results suggest that the new embedding methods consistently outperform the traditional methods by a non-trivial margin on many similarity-oriented tasks. However, state-of-the-art embedding methods are all based on the same bag-of-contexts representation of words. Furthermore, analysis by Levy and Goldberg (2014c) shows that `word2vec`'s SGNS is implicitly factorizing a word-context PMI matrix. That is, the mathematical objective and the sources of information available to SGNS are in fact very similar to those employed by the more traditional methods.

What, then, is the source of superiority (or perceived superiority) of these recent embeddings?

While the focus of the presentation in the word-embedding literature is on the mathematical model and the objective being optimized, other factors affect the results as well. In particular, embedding algorithms suggest some natural *hyperparameters* that can be tuned; many of which were already tuned to some extent by the algorithms' designers. Some hyperparameters, such as the number of negative samples to use, are clearly marked as tunable. Other modifications, such as smoothing the negative-sampling distribution, are reported in passing and considered thereafter as part of the algorithm. Others still, such as dynamically-sized context windows, are not even mentioned in some of the papers, but are part of the canonical implementation. All of these modifications and system design choices, which we collectively denote as *hyperparameters*, are part of the final algorithm, and, as we show, have a substantial impact on performance.

Transactions of the Association for Computational Linguistics, vol. 3, pp. 211–225, 2015. Action Editor: Patrick Pantel.
Submission batch: 1/2015; Revision batch 3/2015; Published 5/2015.

In this work, we make these hyperparameters explicit, and show how they can be *adapted and transferred* into the traditional count-based approach. To asses how each hyperparameter contributes to the algorithms' performance, we conduct a comprehensive set of experiments and compare four different representation methods, while controlling for the various hyperparameters.

Once adapted across methods, hyperparameter tuning significantly improves performance in every task. In many cases, changing the setting of a single hyperparameter yields a greater increase in performance than switching to a better algorithm or training on a larger corpus.

In particular, `word2vec`'s smoothing of the negative sampling distribution can be adapted to PPMI-based methods by introducing a novel, smoothed variant of the PMI association measure (see Section 3.2). Using this variant increases performance by over 3 points per task, on average. We suspect that this smoothing partially addresses the "Achilles' heel" of PMI: its bias towards co-occurrences of rare words.

We also show that when all methods are allowed to tune a similar set of hyperparameters, their performance is largely comparable. In fact, there is no consistent advantage to one algorithmic approach over another, a result that contradicts the claim that embeddings are superior to count-based methods.

2 Background

We consider four word representation methods: the explicit PPMI matrix, SVD factorization of said matrix, SGNS, and GloVe. For historical reasons, we refer to PPMI and SVD as "count-based" representations, as opposed to SGNS and GloVe, which are often referred to as "neural" or "prediction-based" embeddings. All of these methods (as well as all other "skip-gram"-based embedding methods) are essentially bag-of-words models, in which the representation of each word reflects a weighted bag of context-words that co-occur with it. Such bag-of-word embedding models were previously shown to perform as well as or better than more complex embedding methods on similarity and analogy tasks (Mikolov et al., 2013a; Pennington et al., 2014).

Notation We assume a collection of words $w \in V_W$ and their contexts $c \in V_C$, where V_W and V_C are the word and context vocabularies, and denote the collection of observed word-context pairs as D. We use $\#(w, c)$ to denote the number of times the pair (w, c) appears in D. Similarly, $\#(w) = \sum_{c' \in V_C} \#(w, c')$ and $\#(c) = \sum_{w' \in V_W} \#(w', c)$ are the number of times w and c occurred in D, respectively. In some algorithms, words and contexts are embedded in a space of d dimensions. In these cases, each word $w \in V_W$ is associated with a vector $\vec{w} \in \mathbb{R}^d$ and similarly each context $c \in V_C$ is represented as a vector $\vec{c} \in \mathbb{R}^d$. We sometimes refer to the vectors $\vec{w}$ as rows in a $|V_W| \times d$ matrix W, and to the vectors $\vec{c}$ as rows in a $|V_C| \times d$ matrix C. When referring to embeddings produced by a specific method x, we may use W^x and C^x (e.g. W^{SGNS} or C^{SVD}). All vectors are normalized to unit length before they are used for similarity calculation, making cosine similarity and dot-product equivalent (see Section 3.3 for further discussion).

Contexts D is commonly obtained by taking a corpus $w_1, w_2, \ldots, w_n$ and defining the contexts of word w_i as the words surrounding it in an L-sized window $w_{i-L}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+L}$. While other definitions of contexts have been studied (Padó and Lapata, 2007; Baroni and Lenci, 2010; Levy and Goldberg, 2014a) this work focuses on fixed-window bag-of-words contexts.

2.1 Explicit Representations (PPMI Matrix)

The traditional way to represent words in the distributional approach is to construct a high-dimensional sparse matrix M, where each row represents a word w in the vocabulary V_W and each column a potential context $c \in V_C$. The value of each matrix cell M_{ij} represents the association between the word w_i and the context c_j. A popular measure of this association is pointwise mutual information (PMI) (Church and Hanks, 1990). PMI is defined as the log ratio between w and c's joint probability and the product of their marginal probabilities, which can be estimated by:

$$PMI(w, c) = \log \frac{\hat{P}(w,c)}{\hat{P}(w)\hat{P}(c)} = \log \frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)}$$

The rows of M^{PMI} contain many entries of word-context pairs (w, c) that were never observed in the corpus, for which $PMI(w, c) = \log 0 = -\infty$. A common approach is thus to replace the M^{PMI} matrix with M_0^{PMI}, in which $PMI(w, c) = 0$ in cases where $\#(w, c) = 0$. A more consistent approach is to use *positive PMI* (PPMI), in which all

negative values are replaced by 0:

$$PPMI(w,c) = \max\left(PMI\left(w,c\right),0\right)$$

Bullinaria and Levy (2007) showed that M^{PPMI} outperforms M_0^{PMI} on semantic similarity tasks.

A well-known shortcoming of PMI, which persists in PPMI, is its bias towards infrequent events (Turney and Pantel, 2010). A rare context c that co-occurred with a target word w even once, will often yield relatively high PMI score because $\hat{P}(c)$, which is in PMI's denominator, is very small. This creates a situation in which the top "distributional features" (contexts) of w are often extremely rare words, which do not necessarily appear in the respective representations of words that are semantically similar to w. Nevertheless, the PPMI measure is widely regarded as state-of-the-art for these kinds of distributional-similarity models.

2.2 Singular Value Decomposition (SVD)

While sparse vector representations work well, there are also advantages to working with dense low-dimensional vectors, such as improved computational efficiency and, arguably, better generalization. Such vectors can be obtained by performing dimensionality reduction over the sparse high-dimensional matrix.

A common method of doing so is truncated Singular Value Decomposition (SVD), which finds the optimal rank d factorization with respect to L_2 loss (Eckart and Young, 1936). It was popularized in NLP via Latent Semantic Analysis (LSA) (Deerwester et al., 1990).

SVD factorizes M into the product of three matrices $U \cdot \Sigma \cdot V^\top$, where U and V are orthonormal and Σ is a diagonal matrix of eigenvalues in decreasing order. By keeping only the top d elements of Σ, we obtain $M_d = U_d \cdot \Sigma_d \cdot V_d^\top$. The dot-products between the rows of $W = U_d \cdot \Sigma_d$ are equal to the dot-products between rows of M_d.

In the setting of word-context matrices, the dense, d-dimensional rows of W can substitute the very high-dimensional rows of M. Indeed, a common approach in NLP literature is factorizing the PPMI matrix M^{PPMI} with SVD, and then taking the rows of:

$$W^{\text{SVD}} = U_d \cdot \Sigma_d \qquad C^{\text{SVD}} = V_d \qquad (1)$$

as word and context representations, respectively.

2.3 Skip-Grams with Negative Sampling (SGNS)

We present a brief sketch of SGNS – the skip-gram embedding model introduced in (Mikolov et al., 2013a) trained using the negative-sampling procedure presented in (Mikolov et al., 2013b). A detailed derivation of SGNS is available in (Goldberg and Levy, 2014).

SGNS seeks to represent each word $w \in V_W$ and each context $c \in V_C$ as d-dimensional vectors $\vec{w}$ and $\vec{c}$, such that words that are "similar" to each other will have similar vector representations. It does so by trying to maximize a function of the product $\vec{w} \cdot \vec{c}$ for (w,c) pairs that occur in D, and minimize it for negative examples: (w,c_N) pairs that do not necessarily occur in D. The negative examples are created by stochastically corrupting observed (w,c) pairs from D – hence the name "negative sampling". For each observation of (w,c), SGNS draws k contexts from the empirical unigram distribution $P_D(c) = \frac{\#(c)}{|D|}$. In `word2vec`'s implementation of SGNS, this distribution is smoothed, a design choice that boosts its performance. We explore this hyperparameter and others in Section 3.

SGNS as Implicit Matrix Factorization Levy and Golberg (2014c) show that SGNS's corpus-level objective achieves its optimal value when:

$$\vec{w} \cdot \vec{c} = \text{PMI}(w,c) - \log k$$

Hence, SGNS is implicitly factorizing a word-context matrix whose cell's values are PMI, shifted by a global constant ($\log k$):

$$W \cdot C^\top = M^{\text{PMI}} - \log k$$

SGNS performs a different kind of factorization from traditional SVD (see 2.2). In particular, the factorization's loss function is *not* based on L_2, and is much less sensitive to extreme and infinite values due to a sigmoid function surrounding $\vec{w} \cdot \vec{c}$. Furthermore, the loss is weighted, causing rare (w,c) pairs to affect the objective much less than frequent ones. Thus, while many cells in M^{PMI} equal $\log 0 = -\infty$, the cost incurred for reconstructing these cells as a small negative value, such as -5 instead of as $-\infty$, is negligible.[1]

[1] The logistic (sigmoidal) objective also curbs very high positive values of PMI. We suspect that this property, along with the weighted factorization property, addresses the aforementioned shortcoming of PMI, i.e. its overweighting of infrequent events.

An additional difference from SVD, which will be explored further in Section 3.3, is that SVD factorizes M into three matrices, two of them orthonormal and one diagonal, while SGNS factorizes M into two unconstrained matrices.

2.4 Global Vectors (GloVe)

GloVe (Pennington et al., 2014) seeks to represent each word $w \in V_W$ and each context $c \in V_C$ as d-dimensional vectors $\vec{w}$ and $\vec{c}$ such that:

$$\vec{w} \cdot \vec{c} + b_w + b_c = \log(\#(w,c)) \quad \forall (w,c) \in D$$

Here, b_w and b_c (scalars) are word/context-specific biases, and are also parameters to be learned in addition to $\vec{w}$ and $\vec{c}$.

GloVe's objective is explicitly defined as a factorization of the log-count matrix, shifted by the entire vocabularies' bias terms:

$$M^{\log(\#(w,c))} \approx W \cdot C^\top + \vec{b^w} + \vec{b^c}$$

Where $\vec{b^w}$ is a $|V_W|$ dimensional row vector and $\vec{b^c}$ is a $|V_C|$ dimensional column vector.

If we were to fix $b_w = \log \#(w)$ and $b_c = \log \#(c)$, this would be almost[2] equivalent to factorizing the PMI matrix shifted by $\log(|D|)$. However, GloVe learns these parameters, giving an extra degree of freedom over SVD and SGNS. The model is fit to minimize a weighted least square loss, giving more weight to frequent (w,c) pairs.[3]

Finally, an important novelty introduced in (Pennington et al., 2014) is that, assuming $V_C = V_W$, one could take the representation of a word w to be $\vec{w} + \vec{c_w}$ where $\vec{c_w}$ is the row corresponding to w in $C^\top$. This may improve results considerably in some circumstances, as we discuss in Sections 3.3 and 6.2.

3 Transferable Hyperparameters

This section presents various hyperparameters implemented in `word2vec` and GloVe, and shows how to adapt and apply them to count-based methods. We divide these into: pre-processing hyperparameters, which affect the algorithms' input data; association metric hyperparameters, which define how word-context interactions are calculated; and post-processing hyperparameters, which modify the resulting word vectors.

[2]GloVe's objective ignores (w,c) pairs that do not co-occur in the training corpus, treating them as missing values. SGNS, on the other hand, does take such pairs into account through the negative sampling procedure.

[3]The weighting formula is another hyper-parameter that could be tuned, but we keep to the default weighting scheme.

3.1 Pre-processing Hyperparameters

All the matrix-based algorithms rely on a collection D of word-context pairs (w,c) as inputs. `word2vec` introduces three novel variations on the way D is collected, which can be easily applied to other methods beyond SGNS.

Dynamic Context Window (`dyn`) The traditional approaches usually use a constant-sized unweighted context window. For instance, if the window size is 5, then a word five tokens apart from the target is treated the same as an adjacent word. Following the intuition that contexts closer to the target are more important, context words can be weighted according to their distance from the focus word. Both GloVe and `word2vec` employ such a weighting scheme, and while less common, this approach was also explored in traditional count-based methods, e.g. (Sahlgren, 2006).

GloVe's implementation weights contexts using the harmonic function, e.g. a context word three tokens away will be counted as $\frac{1}{3}$ of an occurrence. On the other hand, `word2vec`'s implementation is equivalent to weighing by the distance from the focus word divided by the window size. For example, a size-5 window will weigh its contexts by $\frac{5}{5}, \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5}$.

The reason we call this modification *dynamic* context windows is because `word2vec` implements its weighting scheme by uniformly sampling the actual window size between 1 and L, for each token (Mikolov et al., 2013a). The sampling method is faster than the direct method in terms of training time, since there are fewer SGD updates in SGNS and fewer non-zero matrix cells in the other methods. For our systematic experiments, we used the `word2vec`-style sampled version for all methods, including GloVe.

Subsampling (`sub`) Subsampling is a method of diluting very frequent words, akin to removing stop-words. The subsampling method presented in (Mikolov et al., 2013a) randomly removes words that are more frequent than some threshold t with a probability of p, where f marks the word's corpus frequency:

$$p = 1 - \sqrt{\frac{t}{f}} \tag{2}$$

Following the recommendation in (Mikolov et al., 2013a), we use $t = 10^{-5}$ in our experiments.[4]

[4]`word2vec`'s code implements a slightly different formula: $p = \frac{f-t}{f} - \sqrt{\frac{t}{f}}$. We followed the formula presented

Another implementation detail of subsampling in `word2vec` is that the removal of tokens is done *before* the corpus is processed into word-context pairs. This practically enlarges the context window's size for many tokens, because they can now reach words that were not in their original L-sized windows. We call this kind of subsampling "dirty", as opposed to "clean" subsampling, which removes subsampled words without affecting the context window's size. We found their impact on performance comparable, and report results of only the "dirty" variant.

Deleting Rare Words (`del`) While it is common to ignore words that are rare in the training corpus, `word2vec` removes these tokens from the corpus *before* creating context windows. As with subsampling, this variation narrows the distance between tokens, inserting new word-context pairs that did not exist in the original corpus with the same window size. Though this variation may also have an effect on performance, preliminary experiments showed that it was small, and we therefore do not investigate its effect in this paper.

3.2 Association Metric Hyperparameters

The PMI (or PPMI) between a word and its context is well known to be an effective association measure in the word similarity literature. Levy and Golberg (2014c) show that SGNS is implicitly factorizing a word-context matrix whose cell's values are shifted PMI. Following their analysis, we present two variations of the PMI (and implicitly PPMI) association metric, which we adopt from SGNS. These enhancements of PMI are not directly applicable to GloVe, which, by definition, uses a different association measure.

Shifted PMI (`neg`) SGNS has a natural hyperparameter k (the number of negative samples), which affects the value that SGNS is trying to optimize for each (w, c): $PMI(w, c) - \log k$. The shift caused by $k > 1$ can be applied to distributional methods through shifted PPMI (Levy and Goldberg, 2014c):

$$SPPMI(w, c) = \max\left(PMI\left(w, c\right) - \log k, 0\right)$$

It is important to understand that in SGNS, k has two distinct functions. First, it is used to better estimate the distribution of negative examples; a higher k means more data and better estimation. Second, it acts as a prior on the probability of observing a positive example (an actual occurrence of (w, c) in the corpus) versus a negative example; a higher k means that negative examples are more probable. Shifted PPMI captures only the second aspect of k (a prior). We experiment with three values of k: 1, 5, 15.

Context Distribution Smoothing (`cds`) In `word2vec`, negative examples (contexts) are sampled according to a *smoothed* unigram distribution. In order to smooth the original contexts' distribution, all context counts are raised to the power of α (Mikolov et al. (2013b) found $\alpha = 0.75$ to work well). This smoothing variation has an analog when calculating PMI directly:

$$PMI_\alpha\left(w, c\right) = \log \frac{\hat{P}(w, c)}{\hat{P}(w)\hat{P}_\alpha(c)} \qquad (3)$$

$$\hat{P}_\alpha(c) = \frac{\#\left(c\right)^\alpha}{\sum_c \#\left(c\right)^\alpha}$$

Like other smoothing techniques (Pantel and Lin, 2002; Turney and Littman, 2003), context distribution smoothing alleviates PMI's bias towards rare words. It does so by enlarging the probability of sampling a rare context (since $\hat{P}_\alpha(c) > \hat{P}(c)$ when c is infrequent), which in turn reduces the PMI of (w, c) for any w co-occurring with the rare context c. In Section 6.2 we demonstrate that this novel variant of PMI is very effective, and consistently improves performance across tasks, methods, and configurations. We experiment with two values of α: 1 (unsmoothed) and 0.75 (smoothed).

3.3 Post-processing Hyperparameters

We present three hyperparameters that modify the algorithms' output: the word vectors.

Adding Context Vectors (`w+c`) Pennington et al. (2014) propose using the context vectors in addition to the word vectors as GloVe's output. For example, the word "cat" can be represented as:

$$\vec{v}_{\text{cat}} = \vec{w}_{\text{cat}} + \vec{c}_{\text{cat}}$$

where $\vec{w}$ and $\vec{c}$ are the word and context embeddings, respectively.

This vector combination was originally motivated as an ensemble method. Here, we provide a different interpretation of its effect on the cosine similarity function. Specifically, we show

in the original paper (equation 2).

that adding context vectors effectively adds first-order similarity terms to the second-order similarity function.

Consider the cosine similarity of two words:

$$cos(x,y) = \frac{\vec{v}_x \cdot \vec{v}_y}{\sqrt{\vec{v}_x \cdot \vec{v}_x}\sqrt{\vec{v}_y \cdot \vec{v}_y}} =$$

$$\frac{(\vec{w}_x + \vec{c}_x) \cdot (\vec{w}_y + \vec{c}_y)}{\sqrt{(\vec{w}_x + \vec{c}_x) \cdot (\vec{w}_x + \vec{c}_x)}\sqrt{(\vec{w}_y + \vec{c}_y) \cdot (\vec{w}_y + \vec{c}_y)}}$$

$$= \frac{\vec{w}_x \cdot \vec{w}_y + \vec{c}_x \cdot \vec{c}_y + \vec{w}_x \cdot \vec{c}_y + \vec{c}_x \cdot \vec{w}_y}{\sqrt{\vec{w}_x^2 + 2\vec{w}_x \cdot \vec{c}_x + \vec{c}_x^2}\sqrt{\vec{w}_y^2 + 2\vec{w}_y \cdot \vec{c}_y + \vec{c}_y^2}}$$

$$= \frac{\vec{w}_x \cdot \vec{w}_y + \vec{c}_x \cdot \vec{c}_y + \vec{w}_x \cdot \vec{c}_y + \vec{c}_x \cdot \vec{w}_y}{2\sqrt{\vec{w}_x \cdot \vec{c}_x + 1}\sqrt{\vec{w}_y \cdot \vec{c}_y + 1}} \quad (4)$$

(The last step follows because, as noted in Section 2, the word and context vectors are normalized after training.)

The resulting expression combines similarity terms which can be divided into two groups: second-order similarity ($w_x \cdot w_y$, $c_x \cdot c_y$) and first-order similarity ($w_* \cdot c_*$). The second-order terms measure the extent to which the two words are *replaceable* based on their tendencies to appear in similar contexts, and are the manifestation of Harris's (1954) distributional hypothesis. The first-order terms measure the tendency of one word to appear in the context of the other.

In SVD and SGNS, the first-order similarity terms between w and c converge to $PMI(w,c)$, while in GloVe it converges into their log-count (with some bias terms).

The similarity calculated in equation 4 is thus a symmetric combination of the first-order and second order similarities of x and y, normalized by a function of their reflective first-order similarities:

$$sim(x,y) = \frac{sim_2(x,y) + sim_1(x,y)}{\sqrt{sim_1(x,x) + 1}\sqrt{sim_1(y,y) + 1}}$$

This similarity measure states that words are similar if they tend to appear in similar contexts, *or* if they tend to appear in the contexts of each other (and preferably both).

The additive `w+c` representation can be trivially applied to other methods that produce distinct word and context vectors (e.g. SVD and SGNS). On the other hand, explicit methods such as PPMI are sparse by definition, and nullify the vast majority of first-order similarities. We therefore do not apply `w+c` to PPMI in this study.

Eigenvalue Weighting (`eig`) As mentioned in Section 2.2, the word and context vectors derived using SVD are typically represented by (equation 1):

$$W^{\text{SVD}} = U_d \cdot \Sigma_d \qquad C^{\text{SVD}} = V_d$$

However, this is not necessarily the optimal construction of W^{SVD} for word similarity tasks. We note that in the SVD-based factorization, the resulting word and context matrices have very different properties. In particular, the context matrix C^{SVD} is orthonormal while the word matrix W^{SVD} is not. On the other hand, the factorization achieved by SGNS's training procedure is much more "symmetric", in the sense that neither W^{W2V} nor C^{W2V} is orthonormal, and no particular bias is given to either of the matrices in the training objective. Similar symmetry can be achieved with the following factorization:

$$W = U_d \cdot \sqrt{\Sigma_d} \qquad C = V_d \cdot \sqrt{\Sigma_d} \quad (5)$$

Alternatively, the eigenvalue matrix can be dismissed altogether:

$$W = U_d \qquad C = V_d \quad (6)$$

While it is not theoretically clear why the symmetric approach is better for semantic tasks, it does work much better empirically (see Section 6.1). A similar observation was made by Caron (2001), who suggested adding a parameter p to control the eigenvalue matrix Σ:

$$W^{\text{SVD}_p} = U_d \cdot \Sigma_d^p$$

Later studies show that weighting the eigenvalue matrix Σ_d with the exponent p can have a significant effect on performance, and should be tuned (Bullinaria and Levy, 2012; Turney, 2012). Adapting the notion of symmetric decomposition from SGNS, this study experiments only with symmetric variants of SVD ($p = 0$, $p = 0.5$; equations (6) and (5)) and the traditional factorization ($p = 1$; equation (1)).

Vector Normalization (`nrm`) As mentioned in Section 2, all vectors (i.e. W's rows) are normalized to unit length (L_2 normalization), rendering the dot product operation equivalent to cosine similarity. This normalization is a hyperparameter setting in itself, and other normalizations are also applicable. The trivial case is using no normalization

Hyper-parameter	Explored Values	Applicable Methods
`win`	2, 5, 10	All
`dyn`	none, with	All
`sub`	none, dirty, clean†	All
`del`	none, with†	All
`neg`	1, 5, 15	PPMI, SVD, SGNS
`cds`	1, 0.75	PPMI, SVD, SGNS
`w+c`	only w, $w + c$	SVD, SGNS, GloVe
`eig`	0, 0.5, 1	SVD
`nrm`	none†, row, col†, both†	All

Table 1: The space of hyperparameters explored in this work. †Explored only in preliminary experiments.

at all. Another setting, used by Pennington et al. (2014), normalizes the columns of W rather than its rows. It is also possible to consider a fourth setting that combines both row and column normalizations.

Note that column normalization is akin to dismissing the eigenvalues in SVD. While the hyperparameter setting `eig` $= 0$ has an important positive impact on SVD, the same cannot be said of column normalization on other methods. In preliminary experiments, we tried the four different normalization schemes described above (none, row, column, and both), and found the standard L_2 normalization of W's rows (i.e. using the cosine similarity measure) to be consistently superior.

4 Experimental Setup

We explored a large space of hyperparameters, representations, and evaluation datasets.

4.1 Hyperparameter Space

Table 1 enumerates the hyperparameter space. We generated 72 PPMI, 432 SVD, 144 SGNS, and 24 GloVe representations; 672 overall.

4.2 Word Representations

Corpus All models were trained on English Wikipedia (August 2013 dump), pre-processed by removing non-textual elements, sentence splitting, and tokenization. The corpus contains 77.5 million sentences, spanning 1.5 billion tokens. Models were derived using windows of 2, 5, and 10 tokens to each side of the focus word (the window size parameter is denoted `win`). Words that appeared less than 100 times in the corpus were ignored, resulting in vocabularies of 189,533 terms for both words and contexts.

Training Embeddings We trained a 500-dimensional representation with SVD, SGNS, and GloVe. SGNS was trained using a modified version of `word2vec` which receives a sequence of pre-extracted word-context pairs (Levy and Goldberg, 2014a). GloVe was trained with 50 iterations using the original implementation (Pennington et al., 2014), applied to the pre-extracted word-context pairs.

4.3 Test Datasets

We evaluated each word representation on eight datasets covering similarity and analogy tasks.

Word Similarity We used six datasets to evaluate word similarity: the popular *WordSim353* (Finkelstein et al., 2002) partitioned into two datasets, *WordSim Similarity* and *WordSim Relatedness* (Zesch et al., 2008; Agirre et al., 2009); Bruni et al.'s (2012) *MEN* dataset; Radinsky et al.'s (2011) *Mechanical Turk* dataset; Luong et al.'s (2013) *Rare Words* dataset; and Hill et al.'s (2014) *SimLex-999* dataset. All these datasets contain word pairs together with human-assigned similarity scores. The word vectors are evaluated by ranking the pairs according to their cosine similarities, and measuring the correlation (Spearman's ρ) with the human ratings.

Analogy The two *analogy* datasets present questions of the form "a is to a^* as b is to b^*", where b^* is hidden, and must be guessed from the entire vocabulary. MSR's analogy dataset (Mikolov et al., 2013c) contains 8000 morpho-syntactic analogy questions, such as "*good* is to *best* as *smart* is to *smartest*". Google's analogy dataset (Mikolov et al., 2013a) contains 19544 questions, about half of the same kind as in MSR (syntactic analogies), and another half of a more semantic nature, such as capital cities ("*Paris* is to *France* as *Tokyo* is to *Japan*"). After filtering questions involving out-of-vocabulary words, i.e. words that appeared in English Wikipedia less than 100 times, we remain with 7118 instances in MSR and 19258 instances in Google. The analogy questions are answered using 3CosAdd (addition and subtraction):

$$\arg\max_{b^* \in V_W \setminus \{a^*,b,a\}} \cos(b^*, a^* - a + b) =$$
$$\arg\max_{b^* \in V_W \setminus \{a^*,b,a\}} (\cos(b^*, a^*) - \cos(b^*, a) + \cos(b^*, b))$$

as well as 3CosMul, which is state-of-the-art in analogy recovery (Levy and Goldberg, 2014b):

$$\arg\max_{b^* \in V_W \setminus \{a^*,b,a\}} \frac{\cos(b^*, a^*) \cdot \cos(b^*, b)}{\cos(b^*, a) + \varepsilon}$$

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
PPMI	.709	.540	.688	.648	.393	.338	.491 / **.650**	.246 / .439
SVD	**.776**	**.658**	**.752**	.557	**.506**	**.422**	.452 / .498	.357 / .412
SGNS	.724	.587	.686	**.678**	.434	.401	.530 / .552	.578 / **.592**
GloVe	.666	.467	.659	.599	.403	.398	.442 / .465	.529 / .576

Table 2: Performance of each method across different tasks in the "vanilla" scenario (all hyperparameters set to default): win $= 2$; dyn $=$ none; sub $=$ none; neg $= 1$; cds $= 1$; w+c $=$ only w; eig $= 0.0$.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
PPMI	.755	**.688**	.745	**.686**	.423	.354	.553 / **.629**	.289 / .413
SVD	**.784**	.672	**.777**	.625	**.514**	.402	.547 / .587	.402 / .457
SGNS	.773	.623	.723	.676	.431	**.423**	.599 / .625	.514 / .546
GloVe	.667	.506	.685	.599	.372	.389	.539 / .563	.503 / **.559**
CBOW	.766	.613	.757	.663	.480	.412	.547 / .591	.557 / **.598**

Table 3: Performance of each method across different tasks using word2vec's recommended configuration: win $= 2$; dyn $=$ with; sub $=$ dirty; neg $= 5$; cds $= 0.75$; w+c $=$ only w; eig $= 0.0$. CBOW is presented for comparison.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
PPMI	.755	**.697**	.745	.686	.462	.393	.553 / .679	.306 / .535
SVD	**.793**	.691	**.778**	.666	**.514**	.432	.554 / .591	.408 / .468
SGNS	**.793**	.685	.774	**.693**	.470	**.438**	.676 / **.688**	.618 / **.645**
GloVe	.725	.604	.729	.632	.403	.398	.569 / .596	.533 / .580

Table 4: Performance of each method across different tasks using the best configuration for that method and task combination, assuming win $= 2$.

$\varepsilon = 0.001$ is used to prevent division by zero. We abbreviate the two methods "Add" and "Mul", respectively. The evaluation metric for the analogy questions is the percentage of questions for which the argmax result was the correct answer (b^*).

5 Results

We begin by comparing the effect of various hyperparameter configurations, and observe that different settings have a substantial impact on performance (Section 5.1); at times, this improvement is greater than that of switching to a different representation method. We then show that, in some tasks, careful hyperparameter tuning can also outweigh the importance of adding more data (5.2). Finally, we observe that our results do not agree with a few recent claims in the word embedding literature, and suggest that these discrepancies stem from hyperparameter settings that were not controlled for in previous experiments (5.3).

5.1 Hyperparameters vs Algorithms

We first examine a "vanilla" scenario (Table 2), in which all hyperparameters are "turned off" (set to default values): small context windows (win $=$ 2), no dynamic contexts (dyn $=$ none), no subsampling (sub $=$ none), one negative sample (neg $= 1$), no smoothing (cds $= 1$), no context vectors (w+c $=$ only w), and default eigenvalue weights (eig $= 0.0$).[5] Overall, SVD outperforms other methods on most word similarity tasks, often having a considerable advantage over the second-best. In contrast, analogy tasks present mixed results; SGNS yields the best result in MSR's analogies, while PPMI dominates Google's dataset.

The second scenario (Table 3) sets the hyperparameters to word2vec's default values: small context windows (win $= 2$),[6] dynamic contexts (dyn $=$ with), dirty subsampling (sub $=$ dirty), five negative samples (neg $= 5$), context distribution smoothing (cds $= 0.75$), no context vectors (w+c $=$ only w), and default eigenvalue weights

[5] While it is more common to set eig $= 1$, this setting degrades SVD's performance considerably (see Section 6.1).

[6] While word2vec's default window size is 5, we present a single window size (win $= 2$) in Tables 2-4, in order to isolate win's effect from the effects of other hyperparameters. Running the same experiments with different window sizes reveals similar trends. Additional results with broader window sizes are shown in Table 5.

win	Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
2	PPMI	.732	**.699**	.744	.654	.457	.382	.552 / .677	.306 / .535
	SVD	.772	.671	**.777**	.647	**.508**	.425	.554 / .591	.408 / .468
	SGNS	**.789**	.675	.773	**.661**	.449	**.433**	.676 / **.689**	.617 / **.644**
	GloVe	.720	.605	.728	.606	.389	.388	.649 / .666	.540 / .591
5	PPMI	.732	**.706**	.738	**.668**	.442	.360	.518 / .649	.277 / .467
	SVD	.764	.679	**.776**	.639	**.499**	**.416**	.532 / .569	.369 / .424
	SGNS	**.772**	.690	.772	.663	.454	.403	.692 / **.714**	.605 / **.645**
	GloVe	.745	.617	.746	.631	.416	.389	.700 / .712	.541 / .599
10	PPMI	.735	**.701**	.741	.663	.235	.336	.532 / .605	.249 / .353
	SVD	.766	.681	.770	.628	**.312**	.419	.526 / .562	.356 / .406
	SGNS	**.794**	.700	**.775**	**.678**	.281	**.422**	.694 / .710	.520 / **.557**
	GloVe	.746	.643	.754	.616	.266	.375	.702 / **.712**	.463 / .519
10	SGNS-LS	.766	.681	**.781**	**.689**	**.451**	.414	.739 / **.758**	.690 / **.729**
	GloVe-LS	.678	.624	.752	.639	.361	.371	.732 / .750	.628 / .685

Table 5: Performance of each method across different tasks using 2-fold cross-validation for hyperparameter tuning. Configurations on large-scale (LS) corpora are also presented for comparison.

(eig $= 0.0$). The results in this scenario are quite different than those of the vanilla scenario, with better performance in many cases. However, this change is not uniform, as we observe that different settings boost different algorithms. In fact, the question "Which method is best?" might have a completely different answer when comparing on the same task but with different hyperparameter values. Looking at Table 2 and Table 3, for example, SVD is the best algorithm for SimLex-999 in the vanilla scenario, whereas in the word2vec scenario, it does not perform as well as SGNS.

The third scenario (Table 4) enables the full range of hyperparameters given small context windows (win $= 2$); we evaluate each method on each task given every hyperparameter configuration, and choose the best performance. We see a considerable performance increase across all methods when comparing to both the vanilla (Table 2) and word2vec scenarios (Table 3): the best combination of hyperparameters improves up to 15.7 points beyond the vanilla setting, and over 6 points on average. It appears that selecting the right hyperparameter settings often has more impact than choosing the most suitable algorithm.

Main Result The numbers in Table 4 result from an "oracle" experiment, in which the hyperparameters are tuned on the test data, providing an upper bound on the potential performance improvement of hyperparameter tuning. Are such gains achievable in practice?

Table 5 describes a realistic scenario, where the hyperparameters are tuned on a training set, which is separate from the unseen test data. We also report results for different window sizes (win $= 2, 5, 10$). We use 2-fold cross validation, in which, for each task, the hyperparameters are tuned on each half of the data and evaluated on the other half. The numbers reported in Table 5 are the averages of the two runs for each data-point.

The results indicate that approaching the oracle's improvements are indeed feasible. When comparing the performance of the trained configuration (Table 5) to that of the optimal one (Table 4), their average difference is about 1%, with larger datasets usually finding the optimal configuration. It is therefore both practical and beneficial to properly tune hyperparameters for word similarity and analogy detection tasks.

An interesting observation, which immediately appears when looking at Table 5, is that there is no single method that consistently performs better than the rest. This behavior is visible across all window sizes, and is discussed in further detail in Section 5.3.

5.2 Hyperparameters vs Big Data

An important factor in evaluating distributional methods is the size of corpus and vocabulary, where larger corpora tend to yield better representations. However, training word vectors from larger corpora is more costly in computation time, which could be spent in tuning hyperparameters.

To compare the effect of bigger data versus more flexible hyperparameter settings, we created a large corpus with over 10.5 billion words (7 times larger than our original corpus). This corpus was built from an 8.5 billion word corpus sug-

gested by Mikolov for training word2vec,[7] to which we added UKWaC (Ferraresi et al., 2008). As with the original setup, our vocabulary contained every word that appeared at least 100 times in the corpus, amounting to about 620,000 words. Finally, we fixed the context windows to be broad and dynamic ($\texttt{win} = 10, \texttt{dyn} =$ with), and explored 16 hyperparameter settings comprising of: subsampling (sub), shifted PMI ($\texttt{neg} = 1, 5$), context distribution smoothing (cds), and adding context vectors (w+c). This space is somewhat more restricted than the original hyperparameter space.

In terms of computation, SGNS scales nicely, requiring about half a day of computation per setup. GloVe, on the other hand, took several days to run a single 50-iteration instance for this corpus. Applying the traditional count-based methods to this setting proved technically challenging, as they consumed too much memory to be efficiently manipulated. We thus present results for only SGNS and GloVe (Table 5).

Remarkably, there are some cases (3/6 word similarity tasks) in which tuning a larger space of hyperparameters is indeed more beneficial than expanding the corpus. In other cases, however, more data does seem to pay off, as evident with both analogy tasks.

5.3 Re-evaluating Prior Claims

Prior art raises several claims regarding the superiority of certain methods over the others. However, these studies did not control for the hyperparameters presented in this work. We thus revisit these claims, and examine their validity based on the results in Table 5.[8]

Are embeddings superior to count-based distributional methods? It is commonly believed that modern prediction-based embeddings perform better than traditional count-based methods. This claim was recently supported by a series of systematic evaluations by Baroni et al. (2014). However, our results suggest a different trend. Table 5 shows that in word similarity tasks, the average score of SGNS is actually lower than SVD's when $\texttt{win} = 2, 5$, and it never outperforms SVD by more than 1.7 points in those cases. In Google's analogies SGNS and GloVe indeed perform better than PPMI, but only by a margin of 3.7 points (compare PPMI with $\texttt{win} = 2$ and SGNS with $\texttt{win} = 5$). MSR's analogy dataset is the only case where SGNS and GloVe substantially outperform PPMI and SVD.[9] Overall, there does not seem to be a consistent significant advantage to one approach over the other, thus refuting the claim that prediction-based methods are superior to count-based approaches.

The contradictory results in (Baroni et al., 2014) stem from creating word2vec embeddings with somewhat pre-tuned hyperparameters (recommended by word2vec), and comparing them to "vanilla" PPMI and SVD representations. In particular, shifted PMI (negative sampling) and context distribution smoothing ($\texttt{cds} = 0.75$, equation (3) in Section 3.2) were turned on for SGNS, but not for PPMI and SVD. An additional difference is Baroni et al.'s setting of eig=1, which significantly deteriorates SVD's performance (see Section 6.1).

Is GloVe superior to SGNS? Pennington et al. (2014) show a variety of experiments in which GloVe outperforms SGNS (among other methods). However, our results show the complete opposite. In fact, SGNS outperforms GloVe in *every* task (Table 5). Only when restricted to 3CosAdd, a suboptimal configuration, does GloVe show a 0.8 point advantage over SGNS. This trend persists when scaling up to a larger corpus and vocabulary.

This contradiction can be explained by three major differences in the experimental setup. First, in our experiments, hyperparameters were allowed to vary; in particular, w+c was applied to all the methods, including SGNS. Secondly, Pennington et al. (2014) only evaluated on Google's analogies, but not on MSR's. Finally, in our work, all methods are compared using the same underlying corpus.

It is also important to bear in mind that, by definition, GloVe cannot use two hyperparameters: shifted PMI (neg) and context distribution smoothing (cds). Instead, GloVe learns a set of bias parameters that subsumes these two modifications and many other potential changes to the PMI metric. Albeit its greater flexibility, GloVe does not fair better than SGNS in our experiments.

[7] http://word2vec.googlecode.com/svn/trunk/demo-train-big-model-v1.sh

[8] We note that all conclusions drawn in this section rely on the specific data and settings with which we experiment. It is indeed feasible that experiments on different tasks, data, and hyperparameters may yield other conclusions.

[9] Unlike PPMI, SVD underperforms in both analogy tasks.

Is PPMI on-par with SGNS on analogy tasks? Levy and Goldberg (2014b) show that PPMI and SGNS perform similarly on both Google's and MSR's analogy tasks. Nevertheless, the results in Table 5 show a clear advantage to SGNS. While the gap on Google's analogies is not very large (PPMI lags behind SGNS by only 3.7 points), SGNS consistently outperforms PPMI by a large margin on the MSR dataset. MSR's analogy dataset captures syntactic relations, such as singular-plural inflections for nouns and tense modifications for verbs. We conjecture that capturing these syntactic relations may rely on certain types of contexts, such as determiners and function words, which SGNS might be better at capturing – perhaps due to the way it assigns weights to different examples, or because it also captures negative correlations which are filtered by PPMI.

A deeper look into Levy and Goldberg's (2014b) experiments reveals the use of PPMI with *positional* contexts (i.e. each context is a conjunction of a word and its relative position to the target word), whereas SGNS was employed with regular bag-of-words contexts. Positional contexts might contain relevant information for recovering syntactic analogies, explaining PPMI's relatively high score on MSR's analogy task in (Levy and Goldberg, 2014b).

Does 3CosMul recover more analogies than 3CosAdd? Levy and Goldberg (2014b) show that using similarity multiplication (3CosMul) rather than addition (3CosAdd) improves results on all methods and on every task. This claim is consistent with our findings; indeed, 3CosMul dominates 3CosAdd in every case. The improvement is particularly noticeable for SVD and PPMI, which considerably underperform other methods when using 3CosAdd.

5.4 Comparison with CBOW

Another algorithm featured in `word2vec` is CBOW. Unlike the other methods, CBOW *cannot* be easily expressed as a factorization of a word-context matrix; it ties together the tokens of each context window by representing the context vector as the sum of its words' vectors. It is thus more expressive than the other methods, and has a potential of deriving better word representations.

While Mikolov et al. (2013b) found SGNS to outperform CBOW, Baroni et al. (2014) reports that CBOW had a slight advantage. We compared CBOW to the other methods when setting all the hyperparameters to the defaults provided by `word2vec` (Table 3). With the exception of MSR's analogy task, CBOW is not the best-performing method of any other task in this scenario. Other scenarios showed similar trends in our preliminary experiments.

While CBOW can potentially derive better representations by combining the tokens in each context window, this potential is not realized in practice. Nevertheless, Melamud et al. (2014) show that capturing joint contexts can indeed improve performance on word similarity tasks, and we believe it is a direction worth pursuing.

win	eig	Average Performance
2	0	.612
	0.5	.611
	1	.551
5	0	.616
	0.5	.612
	1	.534
10	0	.584
	0.5	.567
	1	.484

Table 6: The average performance of SVD on word similarity tasks given different values of `eig`, in the vanilla scenario.

6 Hyperparameter Analysis

We analyze the individual impact of each hyperparameter, and try to characterize the conditions in which a certain setting is beneficial.

6.1 Harmful Configurations

Certain hyperparameter settings might cripple the performance of a certain method. We observe two scenarios in which SVD performs poorly.

SVD does not benefit from shifted PPMI. Setting `neg` > 1 consistently deteriorates SVD's performance. Levy and Goldberg (2014c) made a similar observation, and hypothesized that this is a result of the increasing number of zero-cells, which may cause SVD to prefer a factorization that is very close to the zero matrix. SVD's L_2 objective is unweighted, and it does not distinguish between observed and unobserved matrix cells.

Using SVD "correctly" is bad. The traditional way of representing words with SVD uses the eigenvalue matrix (`eig` $= 1$): $W = U_d \cdot \Sigma_d$. Despite being theoretically well-motivated, this setting leads to very poor results in practice, when compared to other settings (`eig` $= 0.5$ or 0). Table 6 demonstrates this gap.

The drop in average accuracy when setting $\texttt{eig} = 1$ is astounding. The performance gap persists under different hyperparameter settings as well, and drops in performance of over 15 points (absolute) when using $\texttt{eig} = 1$ instead of $\texttt{eig} = 0.5$ or 0 are not uncommon. This setting is one of the main reasons for SVD's inferior results in the study by Baroni et al. (2014), and also the reason we chose to use $\texttt{eig} = 0.5$ as the default setting for SVD in the vanilla scenario.

6.2 Beneficial Configurations

To identify which hyperparameter settings are beneficial, we looked at the best configuration of each method on each task. We then counted the number of times each hyperparameter setting was chosen in these configurations (Table 7). Some trends emerge, such as PPMI and SVD's preference towards shorter context windows[10] ($\texttt{win} = 2$), and that SGNS always prefers numerous negative samples ($\texttt{neg} > 1$).

To get a closer look and isolate the effect of each hyperparameter, we controlled for said hyperparameter, and compared the best configurations given each of the hyperparameter's settings. Table 8 shows the difference between default and non-default settings of each hyperparameter.

While many hyperparameter settings can improve performance, they may also degrade it when chosen incorrectly. For instance, in the case of shifted PMI (neg), SGNS consistently profits from $\texttt{neg} > 1$, while SVD's performance is dramatically reduced. For PPMI, the utility of applying $\texttt{neg} > 1$ depends on the type of task: word similarity or analogy. Another example is dynamic context windows (dyn), which is beneficial for MSR's analogy task, but largely detrimental to other tasks.

It appears that the only hyperparameter that can be "blindly" applied in any situation is context distribution smoothing ($\texttt{cds} = 0.75$), yielding a consistent improvement at an insignificant risk. Note that cds helps PPMI more than it does other methods; we suggest that this is because it reduces the relative impact of rare words on the distributional representation, thus addressing PMI's "Achilles' heel".

[10] This might also relate to PMI's bias towards infrequent events (see Section 2.1). Broader windows create more random co-occurrences with rare words, "polluting" the distributional vector with random words that have high PMI scores.

7 Practical Recommendations

It is generally advisable to tune all hyperparameters, as well as algorithm-specific hyperparameters, for the task at hand. However, this may be computationally expensive. We thus provide some "rules of thumb", which we found to work well in our setting:

• Always use context distribution smoothing ($\texttt{cds} = 0.75$) to modify PMI, as described in Section 3.2. It consistently improves performance, and is applicable to PPMI, SVD, and SGNS.

• Do not use SVD "correctly" ($\texttt{eig} = 1$). Instead, use one of the symmetric variants (Section 3.3).

• SGNS is a robust baseline. While it might not be the best method for every task, it does not significantly underperform in any scenario. Moreover, SGNS is the fastest method to train, and cheapest (by far) in terms of disk space and memory consumption.

• With SGNS, prefer many negative samples.

• for both SGNS and GloVe, it is worthwhile to experiment with the $\vec{w} + \vec{c}$ variant, which is cheap to apply (does not require retraining) and can result in substantial gains (as well as substantial losses).

8 Conclusions

Recent embedding methods introduce a plethora of design choices beyond network architecture and optimization algorithms. We reveal that these seemingly minor variations can have a large impact on the success of word representation methods. By showing how to adapt and tune these hyperparameters in traditional methods, we allow a proper comparison between representations, and challenge various claims of superiority from the word embedding literature.

This study also exposes the need for more controlled-variable experiments, and extending the concept of "variable" from the obvious task, data, and method to the often ignored preprocessing steps and hyperparameter settings. We also stress the need for transparent and reproducible experiments, and commend authors such as Mikolov, Pennington, and others for making their code publicly available. In this spirit, we make our code available as well.[11]

[11] http://bitbucket.org/omerlevy/hyperwords

Method	win 2 : 5 : 10	dyn none : with	sub none : dirty	neg 1 : 5 : 15	cds 1.00 : 0.75	w+c only w : $w + c$
PPMI	7 : 1 : 0	4 : 4	4 : 4	2 : 6 : 0	1 : 7	—
SVD	7 : 1 : 0	4 : 4	1 : 7	8 : 0 : 0	2 : 6	7 : 1
SGNS	2 : 3 : 3	6 : 2	4 : 4	0 : 4 : 4	3 : 5	4 : 4
GloVe	1 : 3 : 4	6 : 2	7 : 1	—	—	4 : 4

Table 7: The impact of each hyperparameter, measured by the number of tasks in which the best configuration had that hyperparameter setting. Non-applicable combinations are marked by "—".

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Mul	MSR Mul
PPMI	**+0.5%**	–1.0%	0.0%	**+0.1%**	**+0.4%**	–0.1%	–0.1%	**+1.2%**
SVD	–0.8%	–0.2%	0.0%	**+0.6%**	**+0.4%**	–0.1%	**+0.6%**	**+2.1%**
SGNS	–0.9%	–1.5%	–0.3%	**+0.1%**	–0.1%	–0.1%	–1.0%	**+0.7%**
GloVe	–0.8%	–1.2%	–0.9%	–0.8%	**+0.1%**	–0.9%	–3.3%	**+1.8%**

(a) Performance difference between best models with dyn = *with* and dyn = *none*.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Mul	MSR Mul
PPMI	**+0.6%**	**+1.9%**	**+1.3%**	**+1.0%**	–3.8%	–3.9%	–5.0%	–12.2%
SVD	**+0.7%**	**+0.2%**	**+0.6%**	**+0.7%**	**+0.8%**	–0.3%	**+4.0%**	**+2.4%**
SGNS	**+1.5%**	**+2.2%**	**+1.5%**	**+0.1%**	–0.4%	–0.1%	–4.4%	–5.4%
GloVe	**+0.2%**	–1.3%	–1.0%	–0.2%	–3.4%	–0.9%	–3.0%	–3.6%

(b) Performance difference between best models with sub = *dirty* and sub = *none*.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Mul	MSR Mul
PPMI	**+0.6%**	**+4.9%**	**+1.3%**	**+1.0%**	**+2.2%**	**+0.8%**	–6.2%	–9.2%
SVD	–1.7%	–2.2%	–1.9%	–4.6%	–3.4%	–3.5%	–13.9%	–14.9%
SGNS	**+1.5%**	**+2.9%**	**+2.3%**	**+0.5%**	**+1.5%**	**+1.1%**	**+3.3%**	**+2.1%**
GloVe	—	—	—	—	—	—	—	—

(c) Performance difference between best models with neg > 1 and neg $= 1$.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Mul	MSR Mul
PPMI	**+1.3%**	**+2.8%**	0.0%	**+2.1%**	**+3.5%**	**+2.9%**	**+2.7%**	**+9.2%**
SVD	**+0.4%**	–0.2%	**+0.1%**	**+1.1%**	**+0.4%**	–0.3%	**+1.4%**	**+2.2%**
SGNS	**+0.4%**	**+1.4%**	0.0%	**+0.1%**	0.0%	**+0.2%**	**+0.6%**	0.0%
GloVe	—	—	—	—	—	—	—	—

(d) Performance difference between best models with cds $= 0.75$ and cds $= 1$.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Mul	MSR Mul
PPMI	—	—	—	—	—	—	—	—
SVD	–0.6%	–0.2%	–0.4%	–2.1%	–0.7%	**+0.7%**	–1.8%	–3.4%
SGNS	**+1.4%**	**+2.2%**	**+1.2%**	**+1.1%**	–0.3%	–2.3%	–1.0%	–7.5%
GloVe	**+2.3%**	**+4.7%**	**+3.0%**	–0.1%	–0.7%	–2.6%	**+3.3%**	–8.9%

(e) Performance difference between best models with w+c $= w + c$ and w+c $=$ *only* w.

Table 8: The added value versus the risk of setting each hyperparameter. The figures show the differences in performance between the best achievable configurations when restricting a hyperparameter to different values. This difference indicates the potential gain of tuning a given hyperparameter, as well as the risks of decreased performance when not tuning it. For example, an entry of +9.2% in Table (d) means that the best model with cds $= 0.75$ is 9.2% more accurate (absolute) than the best model with cds $= 1$; i.e. on MSR's analogies, using cds $= 0.75$ instead of cds $= 1$ improved PPMI's accuracy from .443 to .535.

Acknowledgements

This work was supported by the Google Research Award Program and the German Research Foundation via the German-Israeli Project Cooperation (grant DA 1600/1-1). We thank Marco Baroni and Jeffrey Pennington for their valuable comments.

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Boulder, Colorado, June. Association for Computational Linguistics.

Marco Baroni and Alessandro Lenci. 2010. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4):673–721.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Dont count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June. Association for Computational Linguistics.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Elia Bruni, Gemma Boleda, Marco Baroni, and Nam Khanh Tran. 2012. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145, Jeju Island, Korea, July. Association for Computational Linguistics.

John A Bullinaria and Joseph P Levy. 2007. Extracting semantic representations from word co-occurrence statistics: a computational study. *Behavior Research Methods*, 39(3):510–526.

John A Bullinaria and Joseph P Levy. 2012. Extracting semantic representations from word co-occurrence statistics: Stop-lists, stemming, and SVD. *Behavior Research Methods*, 44(3):890–907.

John Caron. 2001. Experiments with LSA scoring: optimal rank and basis. In *Proceedings of the SIAM Computational Information Retrieval Workshop*, pages 157–169.

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407.

C Eckart and G Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218.

Roi Reichart Felix Hill and Anna Korhonen. 2014. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *arXiv preprint arXiv:1408.3456*.

Adriano Ferraresi, Eros Zanchetta, Marco Baroni, and Silvia Bernardini. 2008. Introducing and evaluating ukwac, a very large web-derived corpus of English. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4)*, pages 47–54.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2002. Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, 20(1):116–131.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Zellig Harris. 1954. Distributional structure. *Word*, 10(23):146–162.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, Baltimore, Maryland.

Omer Levy and Yoav Goldberg. 2014b. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180, Baltimore, Maryland.

Omer Levy and Yoav Goldberg. 2014c. Neural word embeddings as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2177–2185.

Minh-Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, Sofia, Bulgaria, August. Association for Computational Linguistics.

Oren Melamud, Ido Dagan, Jacob Goldberger, Idan Szpektor, and Deniz Yuret. 2014. Probabilistic modeling of joint-context in distributional similarity. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 181–190, Baltimore, Maryland, June. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013c. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

Sebastian Padó and Mirella Lapata. 2007. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199.

Patrick Pantel and Dekang Lin. 2002. Discovering word senses from text. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 613–619. ACM.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October. Association for Computational Linguistics.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A word at a time: Computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346. ACM.

Magnus Sahlgren. 2006. *The Word-Space Model*. Ph.D. thesis, Stockholm University.

Peter D. Turney and Michael L. Littman. 2003. Measuring praise and criticism: Inference of semantic orientation from association. *Transactions on Information Systems*, 21(4):315–346.

Peter D. Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188.

Peter D. Turney. 2012. Domain and function: A dual-space model of semantic relations and compositions. *Journal of Artificial Intelligence Research*, 44:533–585.

Torsten Zesch, Christof Müller, and Iryna Gurevych. 2008. Using wiktionary for computing semantic relatedness. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pages 861–866. AAAI Press.