WordRank: Learning Word Embeddings via Robust Ranking

Shihao Ji
Parallel Computing Lab, Intel
shihao.ji@intel.com

Hyokun Yun
Amazon
yunhyoku@amazon.com

Pinar Yanardag
Purdue University
ypinar@purdue.edu

Shin Matsushima
University of Tokyo
shin_matsushima@mist.i.u-tokyo.ac.jp

S. V. N. Vishwanathan
Univ. of California, Santa Cruz
vishy@ucsc.edu

Abstract

Embedding words in a vector space has gained a lot of attention in recent years. While state-of-the-art methods provide efficient computation of word similarities via a low-dimensional matrix embedding, their motivation is often left unclear. In this paper, we argue that word embedding can be naturally viewed as a *ranking* problem due to the ranking nature of the evaluation metrics. Then, based on this insight, we propose a novel framework *WordRank* that efficiently estimates word representations via robust ranking, in which the attention mechanism and robustness to noise are readily achieved via the DCG-like ranking losses. The performance of *WordRank* is measured in word similarity and word analogy benchmarks, and the results are compared to the state-of-the-art word embedding techniques. Our algorithm is very competitive to the state-of-the- arts on large corpora, while outperforms them by a significant margin when the training set is limited (*i.e.*, sparse and noisy). With 17 million tokens, *WordRank* performs almost as well as existing methods using 7.2 billion tokens on a popular word similarity benchmark. Our multi-node distributed implementation of *WordRank* is publicly available for general usage.

1 Introduction

Embedding words into a vector space, such that *semantic* and *syntactic* regularities between words are preserved, is an important sub-task for many applications of natural language processing. Mikolov et al. (2013a) generated considerable excitement in the machine learning and natural language processing communities by introducing a neural network based model, which they call *word2vec*. It was shown that *word2vec* produces state-of-the-art performance on both word similarity as well as word analogy tasks. The word similarity task is to retrieve words that are similar to a given word. On the other hand, word analogy requires answering queries of the form *a:b;c:?*, where *a, b*, and *c* are words from the vocabulary, and the answer to the query must be semantically related to *c* in the same way as *b* is related to *a*. This is best illustrated with a concrete example: Given the query *king:queen;man:?* we expect the model to output *woman*.

The impressive performance of *word2vec* led to a flurry of papers, which tried to explain and improve the performance of *word2vec* both theoretically (Arora et al., 2015) and empirically (Levy and Goldberg, 2014). One interpretation of *word2vec* is that it is approximately maximizing the positive pointwise mutual information (PMI), and Levy and Goldberg (2014) showed that directly optimizing this gives good results. On the other hand, Pennington et al. (2014) showed performance comparable to *word2vec* by using a modified matrix factorization model, which optimizes a log loss.

Somewhat surprisingly, Levy et al. (2015) showed that much of the performance gains of these new word embedding methods are due to certain hyper-parameter optimizations and system-design choices. In other words, if one sets up careful experiments, then existing word embedding models more or less perform comparably to each other. We conjecture that this is because, at a high level, all these methods

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 658–668,
Austin, Texas, November 1-5, 2016.

are based on the following template: From a large text corpus eliminate infrequent words, and compute a $|\mathcal{W}| \times |\mathcal{C}|$ word-context co-occurrence count matrix; a context is a word which appears less than d distance away from a given word in the text, where d is a tunable parameter. Let $w \in \mathcal{W}$ be a word and $c \in \mathcal{C}$ be a context, and let $X_{w,c}$ be the (potentially normalized) co-occurrence count. One learns a function $f(w, c)$ which approximates a transformed version of $X_{w,c}$. Different methods differ essentially in the transformation function they use and the parametric form of f (Levy et al., 2015). For example, *GloVe* (Pennington et al., 2014) uses $f(w, c) = \langle \mathbf{u}_w, \mathbf{v}_c \rangle$ where $\mathbf{u}_w$ and $\mathbf{v}_c$ are k dimensional vectors, $\langle \cdot, \cdot \rangle$ denotes the Euclidean dot product, and one approximates $f(w, c) \approx \log X_{w,c}$. On the other hand, as Levy and Goldberg (2014) show, *word2vec* can be seen as using the same $f(w, c)$ as *GloVe* but trying to approximate $f(w, c) \approx \text{PMI}(X_{w,c}) - \log n$, where $\text{PMI}(\cdot)$ is the pairwise mutual information (Cover and Thomas, 1991) and n is the number of negative samples.

In this paper, we approach the word embedding task from a different perspective by formulating it as a *ranking* problem. That is, given a word w, we aim to output an ordered list $(c_1, c_2, \cdots)$ of context words from $\mathcal{C}$ such that words that co-occur with w appear at the top of the list. If $\text{rank}(w, c)$ denotes the rank of c in the list, then typical ranking losses optimize the following objective: $\sum_{(w,c)\in\Omega} \rho\left(\text{rank}(w, c)\right)$, where $\Omega \subset \mathcal{W} \times \mathcal{C}$ is the set of word-context pairs that co-occur in the corpus, and $\rho(\cdot)$ is a ranking loss function that is monotonically increasing and concave (see Sec. 2 for a justification).

Casting word embedding as ranking has two distinctive advantages. First, our method is *discriminative* rather than generative; in other words, instead of modeling (a transformation of) $X_{w,c}$ directly, we only aim to model the relative order of $X_{w,\cdot}$ values in each row. This formulation fits naturally to popular word embedding tasks such as word similarity/analogy since instead of the likelihood of each word, we are interested in finding the most relevant words in a given context[1]. Second, casting word embedding as a ranking problem enables us to design models robust to noise (Yun et al., 2014) and focusing more on differentiating top relevant words, a kind of attention mechanism that has been proved very useful in deep learning (Larochelle and Hinton, 2010; Mnih et al., 2014; Bahdanau et al., 2015). Both issues are very critical in the domain of word embedding since (1) the co-occurrence matrix might be noisy due to grammatical errors or unconventional use of language, *i.e.*, certain words might co-occur purely by chance, a phenomenon more acute in smaller document corpora collected from diverse sources; and (2) it's very challenging to sort out a few most relevant words from a very large vocabulary, thus some kind of attention mechanism that can trade off the resolution on most relevant words with the resolution on less relevant words is needed. We will show in the experiments that our method can mitigate some of these issues; with 17 million tokens our method performs almost as well as existing methods using 7.2 billion tokens on a popular word similarity benchmark.

2 Word Embedding via Ranking

2.1 Notation

We use w to denote a word and c to denote a context. The set of all words, that is, the vocabulary is denoted as $\mathcal{W}$ and the set of all context words is denoted $\mathcal{C}$. We will use $\Omega \subset \mathcal{W} \times \mathcal{C}$ to denote the set of all word-context pairs that were observed in the data, Ω_w to denote the set of contexts that co-occured with a given word w, and similarly Ω_c to denote the words that co-occurred with a given context c. The size of a set is denoted as $|\cdot|$. The inner product between vectors is denoted as $\langle \cdot, \cdot \rangle$.

2.2 Ranking Model

Let $\mathbf{u}_w$ denote the k-dimensional embedding of a word w, and $\mathbf{v}_c$ denote that of a context c. For convenience, we collect embedding parameters for words and contexts as $\mathbf{U} := \{\mathbf{u}_w\}_{w\in\mathcal{W}}$, and $\mathbf{V} := \{\mathbf{v}_c\}_{c\in\mathcal{C}}$.

We aim to capture the relevance of context c for word w by the inner product between their embedding vectors, $\langle \mathbf{u}_w, \mathbf{v}_c \rangle$; the more relevant a context is, the larger we want their inner product to be.

[1] Roughly speaking, this difference in viewpoint is analogous to the difference between pointwise loss function vs listwise loss function used in ranking (Lee and Lin, 2013).

We achieve this by learning a ranking model that is parametrized by $\mathbf{U}$ and $\mathbf{V}$. If we sort the set of contexts $\mathcal{C}$ for a given word w in terms of each context's inner product score with the word, the rank of a specific context c in this list can be written as (Usunier et al., 2009):

$$\begin{aligned}\text{rank}\,(w,c) &= \sum_{c'\in\mathcal{C}\setminus\{c\}} I\left(\langle \mathbf{u}_w, \mathbf{v}_c\rangle - \langle \mathbf{u}_w, \mathbf{v}_{c'}\rangle \leq 0\right) \\ &= \sum_{c'\in\mathcal{C}\setminus\{c\}} I\left(\langle \mathbf{u}_w, \mathbf{v}_c - \mathbf{v}_{c'}\rangle \leq 0\right), \quad (1)\end{aligned}$$

where $I(x \leq 0)$ is a 0-1 loss function which is 1 if $x \leq 0$ and 0 otherwise. Since $I(x \leq 0)$ is a discontinuous function, we follow the popular strategy in machine learning which replaces the 0-1 loss by its convex upper bound $\ell(\cdot)$, where $\ell(\cdot)$ can be any popular loss function for binary classification such as the hinge loss $\ell(x) = \max(0, 1-x)$ or the logistic loss $\ell(x) = \log_2(1+2^{-x})$ (Bartlett et al., 2006). This enables us to construct the following convex upper bound on the rank:

$$\text{rank}\,(w,c) \leq \overline{\text{rank}}\,(w,c) = \sum_{c'\in\mathcal{C}\setminus\{c\}} \ell(\langle \mathbf{u}_w, \mathbf{v}_c - \mathbf{v}_{c'}\rangle) \quad (2)$$

It is certainly desirable that the ranking model positions relevant contexts at the top of the list; this motivates us to write the objective function to minimize as:

$$J(\mathbf{U},\mathbf{V}) := \sum_{w\in\mathcal{W}} \sum_{c\in\Omega_w} r_{w,c} \cdot \rho\left(\frac{\overline{\text{rank}}\,(w,c)+\beta}{\alpha}\right) \quad (3)$$

where $r_{w,c}$ is the weight between word w and context c quantifying the association between them, $\rho(\cdot)$ is a monotonically increasing and concave ranking loss function that measures goodness of a rank, and $\alpha > 0$, $\beta > 0$ are the hyperparameters of the model whose role will be discussed later. Following Pennington et al. (2014), we use

$$r_{w,c} = \begin{cases} (X_{w,c}/x_{max})^{\epsilon} & \text{if } X_{w,c} < x_{max} \\ 1 & \text{otherwise,} \end{cases} \quad (4)$$

where we set $x_{max} = 100$ and $\epsilon = 0.75$ in our experiments. That is, we assign larger weights (with a saturation) to contexts that appear more often with the word of interest, and vice-versa. For the ranking loss function $\rho(\cdot)$, on the other hand, we consider the class of monotonically increasing and concave functions. While monotonicity is a natural requirement, we argue that concavity is also important so that the derivative of ρ is always non-increasing; this implies that the ranking loss to be the most sensitive at the top of the list (where the rank is small) and becomes less sensitive at the lower end of the list (where the rank is high). Intuitively this is desirable, because we are interested in a small number of relevant contexts which frequently co-occur with a given word, and thus are willing to tolerate errors on infrequent contexts[2]. Meanwhile, this insensitivity at the bottom of the list makes the model robust to noise in the data either due to grammatical errors or unconventional use of language. Therefore, a single ranking loss function $\rho(\cdot)$ serves two different purposes at two ends of the curve (see the example plots of ρ in Figure 1); while the left hand side of the curve encourages "high resolution" on most relevant words, the right hand side becomes less sensitive (with "low resolution") to infrequent and possibly noisy words[3]. As we will demonstrate in our experiments, this is a fundamental attribute (in addition to the ranking nature) of our method that contributes its superior performance as compared to the state-of-the-arts when the training set is limited (*i.e.*, sparse and noisy).

What are interesting loss functions that can be used for $\rho(\cdot)$? Here are four possible alternatives, all of which have a natural interpretation (see the plots of all four ρ functions in Figure 1(a) and the related work in Sec. 3 for a discussion).

$$\rho_0(x) := x \quad \text{(identity)} \quad (5)$$

$$\rho_1(x) := \log_2(1+x) \quad \text{(logarithm)} \quad (6)$$

$$\rho_2(x) := 1 - \frac{1}{\log_2(2+x)} \quad \text{(negative DCG)} \quad (7)$$

$$\rho_3(x) := \frac{x^{1-t}-1}{1-t} \quad (\log_t \text{ with } t \neq 1) \quad (8)$$

[2]This is similar to the attention mechanism found in human visual system that is able to focus on a certain region of an image with "high resolution" while perceiving the surrounding image in "low resolution" (Larochelle and Hinton, 2010; Mnih et al., 2014).

[3]Due to the linearity of $\rho_0(x) = x$, this ranking loss doesn't have the benefit of attention mechanism and robustness to noise since it treats all ranking errors uniformly.

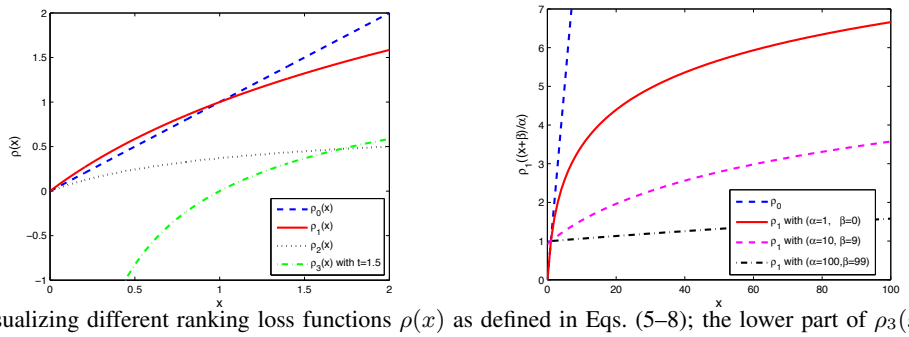

Figure 1: (a) Visualizing different ranking loss functions $\rho(x)$ as defined in Eqs. (5–8); the lower part of $\rho_3(x)$ is truncated in order to visualize the other functions better. (b) Visualizing $\rho_1((x+\beta)/\alpha)$ with different α and β; ρ_0 is included to illustrate the dramatic scale differences between ρ_0 and ρ_1.

We will explore the performance of each of these variants in our experiments. For now, we turn our attention to efficient stochastic optimization of the objective function (3).

2.3 Stochastic Optimization

Plugging (2) into (3), and replacing $\sum_{w\in\mathcal{W}}\sum_{c\in\Omega_w}$ by $\sum_{(w,c)\in\Omega}$, the objective function becomes:

$$J(\mathbf{U},\mathbf{V}) = \sum_{(w,c)\in\Omega} r_{w,c}\cdot \rho\left(\frac{\sum_{c'\in\mathcal{C}\setminus\{c\}}\ell\left(\langle\mathbf{u}_w,\mathbf{v}_c-\mathbf{v}_{c'}\rangle\right)+\beta}{\alpha}\right). \quad (9)$$

This function contains summations over Ω and $\mathcal{C}$, both of which are expensive to compute for a large corpus. Although stochastic gradient descent (SGD) (Bottou and Bousquet, 2011) can be used to replace the summation over Ω by random sampling, the summation over $\mathcal{C}$ cannot be avoided unless $\rho(\cdot)$ is a linear function. To work around this problem, we propose to optimize a linearized upper bound of the objective function obtained through a first-order Taylor approximation. Observe that due to the concavity of $\rho(\cdot)$, we have

$$\rho(x) \le \rho\left(\xi^{-1}\right) + \rho'\left(\xi^{-1}\right)\cdot\left(x-\xi^{-1}\right) \quad (10)$$

for any x and $\xi \neq 0$. Moreover, the bound is tight when $\xi = x^{-1}$. This motivates us to introduce a set of auxiliary parameters $\Xi := \{\xi_{w,c}\}_{(w,c)\in\Omega}$ and define the following upper bound of $J(\mathbf{U},\mathbf{V})$:

$$\overline{J}(\mathbf{U},\mathbf{V},\Xi) := \sum_{(w,c)\in\Omega} r_{w,c}\cdot\left\{\rho(\xi_{wc}^{-1}) + \rho'(\xi_{wc}^{-1})\cdot\left(\alpha^{-1}\beta+\alpha^{-1}\sum_{c'\in\mathcal{C}\setminus\{c\}}\ell\left(\langle\mathbf{u}_w,\mathbf{v}_c-\mathbf{v}_{c'}\rangle\right)-\xi_{w,c}^{-1}\right)\right\}. \quad (11)$$

Note that $J(\mathbf{U},\mathbf{V}) \le \overline{J}(\mathbf{U},\mathbf{V},\Xi)$ for any Ξ, due to (10)[4]. Also, minimizing (11) yields the same $\mathbf{U}$ and $\mathbf{V}$ as minimizing (9). To see this, suppose $\hat{\mathbf{U}} := \{\hat{\mathbf{u}}_w\}_{w\in\mathcal{W}}$ and $\hat{\mathbf{V}} := \{\hat{\mathbf{v}}_c\}_{c\in\mathcal{C}}$ minimizes (9). Then, by letting $\hat{\Xi} := \left\{\hat{\xi}_{w,c}\right\}_{(w,c)\in\Omega}$ where

$$\hat{\xi}_{w,c} = \frac{\alpha}{\sum_{c'\in\mathcal{C}\setminus\{c\}}\ell\left(\langle\hat{\mathbf{u}}_w,\hat{\mathbf{v}}_c-\hat{\mathbf{v}}_{c'}\rangle\right)+\beta}, \quad (12)$$

we have $\overline{J}\left(\hat{\mathbf{U}},\hat{\mathbf{V}},\hat{\Xi}\right) = J\left(\hat{\mathbf{U}},\hat{\mathbf{V}}\right)$. Therefore, it suffices to optimize (11). However, unlike (9), (11) admits an efficient SGD algorithm. To see this, rewrite (11) as

$$\overline{J}(\mathbf{U},\mathbf{V},\Xi) = \sum_{(w,c,c')} r_{w,c}\cdot\left(\frac{\rho(\xi_{w,c}^{-1})+\rho'(\xi_{w,c}^{-1})\cdot(\alpha^{-1}\beta-\xi_{w,c}^{-1})}{|\mathcal{C}|-1} + \frac{1}{\alpha}\rho'(\xi_{w,c}^{-1})\cdot\ell\left(\langle\mathbf{u}_w,\mathbf{v}_c-\mathbf{v}_{c'}\rangle\right)\right), \quad (13)$$

where $(w,c,c') \in \Omega\times(\mathcal{C}\setminus\{c\})$. Then, it can be seen that if we sample uniformly from $(w,c)\in\Omega$ and $c'\in\mathcal{C}\setminus\{c\}$, then $j(w,c,c') :=$

$$|\Omega|\cdot(|\mathcal{C}|-1)\cdot r_{w,c}\cdot\left(\frac{\rho(\xi_{w,c}^{-1})+\rho'(\xi_{w,c}^{-1})\cdot(\alpha^{-1}\beta-\xi_{w,c}^{-1})}{|\mathcal{C}|-1} + \frac{1}{\alpha}\rho'(\xi_{w,c}^{-1})\cdot\ell\left(\langle\mathbf{u}_w,\mathbf{v}_c-\mathbf{v}_{c'}\rangle\right)\right), \quad (14)$$

which does not contain any expensive summations and is an unbiased estimator of (13), i.e., $\mathbb{E}[j(w,c,c')] = \overline{J}(\mathbf{U},\mathbf{V},\Xi)$. On the other hand, one can optimize $\xi_{w,c}$ exactly by using (12). Putting

[4] When $\rho=\rho_0$, one can simply set the auxiliary variables $\xi_{w,c}=1$ because ρ_0 is already a linear function.

everything together yields a stochastic optimization algorithm *WordRank*, which can be specialized to a variety of ranking loss functions $\rho(\cdot)$ with weights $r_{w,c}$ (e.g., DCG (Discounted Cumulative Gain) (Manning et al., 2008) is one of many possible instantiations). Algorithm 1 contains detailed pseudo-code. It can be seen that the algorithm is divided into two stages: a stage that updates $(\mathbf{U}, \mathbf{V})$ and another that updates Ξ. Note that the time complexity of the first stage is $\mathcal{O}(|\Omega|)$ since the cost of each update in Lines 8–10 is independent of the size of the corpus. On the other hand, the time complexity of updating Ξ in Line 15 is $\mathcal{O}(|\Omega|\,|\mathcal{C}|)$, which can be expensive. To amortize this cost, we employ two tricks: we only update Ξ after a few iterations of $\mathbf{U}$ and $\mathbf{V}$ update, and we exploit the fact that the most computationally expensive operation in (12) involves a matrix and matrix multiplication which can be calculated efficiently via the SGEMM routine in BLAS (Dongarra et al., 1990).

Algorithm 1 *WordRank* algorithm.

1: η: step size
2: **repeat**
3: `// Stage 1: Update` $\mathbf{U}$ `and` $\mathbf{V}$
4: **repeat**
5: Sample (w, c) uniformly from Ω
6: Sample c' uniformly from $\mathcal{C} \setminus \{c\}$
7: `// following three updates are executed simultaneously`
8: $\mathbf{u}_w \leftarrow \mathbf{u}_w - \eta \cdot r_{w,c} \cdot \rho'(\xi_{w,c}^{-1}) \cdot \ell'\left(\langle \mathbf{u}_w, \mathbf{v}_c - \mathbf{v}_{c'} \rangle\right) \cdot (\mathbf{v}_c - \mathbf{v}_{c'})$
9: $\mathbf{v}_c \leftarrow \mathbf{v}_c - \eta \cdot r_{w,c} \cdot \rho'(\xi_{w,c}^{-1}) \cdot \ell'\left(\langle \mathbf{u}_w, \mathbf{v}_c - \mathbf{v}_{c'} \rangle\right) \cdot \mathbf{u}_w$
10: $\mathbf{v}_{c'} \leftarrow \mathbf{v}_{c'} + \eta \cdot r_{w,c} \cdot \rho'(\xi_{w,c}^{-1}) \cdot \ell'\left(\langle \mathbf{u}_w, \mathbf{v}_c - \mathbf{v}_{c'} \rangle\right) \cdot \mathbf{u}_w$
11: **until** $\mathbf{U}$ and $\mathbf{V}$ are converged
12: `// Stage 2: Update` Ξ
13: **for** $w \in \mathcal{W}$ **do**
14: **for** $c \in \mathcal{C}$ **do**
15: $\xi_{w,c} = \alpha / \left(\sum_{c' \in \mathcal{C} \setminus \{c\}} \ell\left(\langle \mathbf{u}_w, \mathbf{v}_c - \mathbf{v}_{c'} \rangle\right) + \beta \right)$
16: **end for**
17: **end for**
18: **until** $\mathbf{U}$, $\mathbf{V}$ and Ξ are converged

2.4 Parallelization

The updates in Lines 8–10 have one remarkable property: To update $\mathbf{u}_w$, $\mathbf{v}_c$ and $\mathbf{v}_{c'}$, we only need to *read* the variables $\mathbf{u}_w$, $\mathbf{v}_c$, $\mathbf{v}_{c'}$ and $\xi_{w,c}$. What this means is that updates to another triplet of variables $\mathbf{u}_{\hat{w}}$, $\mathbf{v}_{\hat{c}}$ and $\mathbf{v}_{\hat{c}'}$ can be performed independently. This observation is the key to developing a parallel optimization strategy, by distributing the computation of the updates among multiple processors. Due to lack of space, details including pseudo-code are relegated to the supplementary material.

2.5 Interpreting of α and β

The update (12) indicates that $\xi_{w,c}^{-1}$ is proportional to $\overline{\mathrm{rank}}\,(w, c)$. On the other hand, one can observe that the loss function $\ell\,(\cdot)$ in (14) is weighted by a $\rho'\left(\xi_{w,c}^{-1}\right)$ term. Since $\rho\,(\cdot)$ is concave, its gradient $\rho'\,(\cdot)$ is monotonically non-increasing (Rockafellar, 1970). Consequently, when $\overline{\mathrm{rank}}\,(w, c)$ and hence $\xi_{w,c}^{-1}$ is large, $\rho'\left(\xi_{w,c}^{-1}\right)$ is small. In other words, the loss function "gives up" on contexts with high ranks in order to focus its attention on top of the list. The rate at which the algorithm gives up is determined by the hyperparameters α and β. For the illustration of this effect, see the example plots of ρ_1 with different α and β in Figure 1(b). Intuitively, α can be viewed as a *scale* parameter while β can be viewed as an *offset* parameter. An equivalent interpretation is that by choosing different values of α and β one can modify the behavior of the ranking loss $\rho\,(\cdot)$ in a problem dependent fashion. In our experiments, we found that a common setting of $\alpha = 1$ and $\beta = 0$ often yields uncompetitive performance, while setting $\alpha = 100$ and $\beta = 99$ generally gives good results.

3 Related Work

Our work sits at the intersection of word embedding and ranking optimization. As we discussed in Sec. 2.2 and Sec. 2.5, it's also related to the attention mechanism widely used in deep learning. We therefore review the related work along these three axes.

Word Embedding. We already discussed some related work (*word2vec* and *GloVe*) on word embedding in the introduction. Essentially, *word2vec* and *GloVe* derive word representations by modeling a transformation (PMI or log) of $X_{w,c}$ directly, while

WordRank learns word representations via robust ranking. Besides these state-of-the-art techniques, a few ranking-based approaches have been proposed for word embedding recently, e.g., (Collobert and Weston, 2008; Vilnis and McCallum, 2015; Liu et al., 2015). However, all of them adopt a pair-wise binary classification approach with a linear ranking loss ρ_0. For example, (Collobert and Weston, 2008; Vilnis and McCallum, 2015) employ a hinge loss on positive/negative word pairs to learn word representations and ρ_0 is used *implicitly* to evaluate ranking losses. As we discussed in Sec. 2.2, ρ_0 has no benefit of the attention mechanism and robustness to noise since its linearity treats all the ranking errors uniformly; empirically, sub-optimal performances are often observed with ρ_0 in our experiments. More recently, by extending the Skip-Gram model of *word2vec*, Liu et al. (2015) incorporates additional pair-wise constraints induced from 3rd-party knowledge bases, such as WordNet, and learns word representations jointly. In contrast, *WordRank* is a fully ranking-based approach without using any additional data source for training.

Robust Ranking. The second line of work that is very relevant to *WordRank* is that of ranking objective (3). The use of score functions $\langle \mathbf{u}_w, \mathbf{v}_c \rangle$ for ranking is inspired by the latent collaborative retrieval framework of Weston et al. (2012). Writing the rank as a sum of indicator functions (1), and upper bounding it via a convex loss (2) is due to Usunier et al. (2009). Using $\rho_0(\cdot)$ (5) corresponds to the well-known pairwise ranking loss (see *e.g.*, (Lee and Lin, 2013)). On the other hand, Yun et al. (2014) observed that if they set $\rho = \rho_2$ as in (7), then $-J(\mathbf{U}, \mathbf{V})$ corresponds to the DCG (Discounted Cumulative Gain), one of the most popular ranking metrics used in web search ranking (Manning et al., 2008). In their RobiRank algorithm they proposed the use of $\rho = \rho_1$ (6), which they considered to be a special function for which one can derive an efficient stochastic optimization procedure. However, as we showed in this paper, the general class of monotonically increasing concave functions can be handled efficiently. Another important difference of our approach is the hyperparameters α and β, which we use to modify the behavior of ρ, and which we find are critical to achieve good empirical results. Ding and Vishwanathan (2010) proposed the use of $\rho = \log_t$ in the context of robust *binary* classification, while here we are concerned with ranking, and our formulation is very general and applies to a variety of ranking losses $\rho(\cdot)$ with weights $r_{w,c}$. Optimizing over $\mathbf{U}$ and $\mathbf{V}$ by distributing the computation across processors is inspired by work on distributed stochastic gradient for matrix factorization (Gemulla et al., 2011).

Attention. Attention is one of the most important advancements in deep learning in recent years (Larochelle and Hinton, 2010), and is now widely used in state-of-the-art image recognition and machine translation systems (Mnih et al., 2014; Bahdanau et al., 2015). Recently, attention has also been applied to the domain of word embedding. For example, under the intuition that not all contexts are created equal, Wang et al. (2015) assign an importance weight to each word type at each context position and learn an attention-based Continuous Bag-Of-Words (CBOW) model. Similarly, within a ranking framework, *WordRank* expresses the context importance by introducing the auxiliary variable $\xi_{w,c}$, which "gives up" on contexts with high ranks in order to focus its attention on top of the list.

4 Experiments

In our experiments, we first evaluate the impact of the weight $r_{w,c}$ and the ranking loss function $\rho(\cdot)$ on the test performance using a small dataset. We then pick the best performing model and compare it against *word2vec* (Mikolov et al., 2013b) and *GloVe* (Pennington et al., 2014). We closely follow the framework of Levy et al. (2015) to set up a careful and fair comparison of the three methods. Our code is publicly available at `https://bitbucket.org/shihaoji/wordrank`.

Training Corpus Models are trained on a combined corpus of 7.2 billion tokens, which consists of the 2015 Wikipedia dump with 1.6 billion tokens, the WMT14 News Crawl[5] with 1.7 billion tokens, the "One Billion Word Language Modeling Benchmark"[6] with almost 1 billion tokens, and UMBC

[5] `http://www.statmt.org/wmt14/translation-task.html`

[6] `http://www.statmt.org/lm-benchmark`

Corpus Size	17M*	32M	64M	128M	256M	512M	1.0B	1.6B	7.2B
Vocabulary Size $\|\mathcal{W}\|$	71K	100K	100K	200K	200K	300K	300K	400K	620K
Window Size win	15	15	15	10	10	10	10	10	10
Dimension k	100	100	100	200	200	300	300	300	300

* This is the Text8 dataset from `http://mattmahoney.net/dc/text8.zip`, which is widely used for word embedding demo.

Table 1: Parameter settings used in the experiments.

Task	Robi	ρ_0		ρ_1		ρ_2		ρ_3	
		off	on	off	on	off	on	off	on
Similarity	41.2	69.0	<u>71.0</u>	66.7	70.4	66.8	70.8	68.1	68.0
Analogy	22.7	24.9	31.9	34.3	<u>44.5</u>	32.3	40.4	33.6	42.9

Table 2: Performance of different ρ functions on Text8 dataset with 17M tokens.

webbase corpus[7] with around 3 billion tokens. The pre-processing pipeline breaks the paragraphs into sentences, tokenizes and lowercases each corpus with the Stanford tokenizer. We further clean up the dataset by removing non-ASCII characters and punctuation, and discard sentences that are shorter than 3 tokens or longer than 500 tokens. In the end, we obtain a dataset of 7.2 billion tokens, with the first 1.6 billion tokens from Wikipedia. When we want to experiment with a smaller corpus, we extract a subset which contains the specified number of tokens.

Co-occurrence matrix construction We use the *GloVe* code to construct the co-occurrence matrix X, and the same matrix is used to train *GloVe* and *WordRank* models. When constructing X, we must choose the size of the vocabulary, the context window and whether to distinguish left context from right context. We follow the findings and design choices of *GloVe* and use a symmetric window of size win with a decreasing weighting function, so that word pairs that are d words apart contribute $1/d$ to the total count. Specifically, when the corpus is small (*e.g.*, 17M, 32M, 64M) we let $win = 15$ and for larger corpora we let $win = 10$. The larger window size alleviates the data sparsity issue for small corpus at the expense of adding more noise to X. The parameter settings used in our experiments are summarized in Table 1.

Using the trained model It has been shown by Pennington et al. (2014) that combining the $\mathbf{u}_w$ and $\mathbf{v}_c$ vectors with equal weights gives a small boost in performance. This vector combination was originally motivated as an ensemble method (Pennington et al., 2014), and later Levy et al. (2015) provided a different interpretation of its effect on the cosine similarity function, and show that adding context vectors effectively adds first-order similarity terms to the second-order similarity function. In our experiments, we find that vector combination boosts the performance in word analogy task when training set is small, but when dataset is large enough (*e.g.*, 7.2 billion tokens), vector combination doesn't help anymore. More interestingly, for the word similarity task, we find that vector combination is detrimental in all the cases, sometimes even substantially[8]. Therefore, we will always use $\mathbf{u}_w$ on word similarity task, and use $\mathbf{u}_w + \mathbf{v}_c$ on word analogy task unless otherwise noted.

4.1 Evaluation

Word Similarity We use six datasets to evaluate word similarity: WS-353 (Finkelstein et al., 2002) partitioned into two subsets: WordSim Similarity and WordSim Relatedness (Agirre et al., 2009); MEN (Bruni et al., 2012); Mechanical Turk (Radinsky et al., 2011); Rare words (Luong et al., 2013); and SimLex-999 (Hill et al., 2014). They contain word pairs together with human-assigned similarity judgments. The word representations are evaluated by ranking the pairs according to their cosine similarities, and measuring the Spearman's rank correlation coefficient with the human judgments.

[7] `http://ebiquity.umbc.edu/resource/html/id/351`

[8] This is possible since we optimize a ranking loss: the absolute scores don't matter as long as they yield an ordered list correctly. Thus, *WordRank*'s $\mathbf{u}_w$ and $\mathbf{v}_c$ are less comparable to each other than those generated by *GloVe*, which employs a point-wise L_2 loss.

Word Analogies For this task, we use the Google analogy dataset (Mikolov et al., 2013a). It contains 19544 word analogy questions, partitioned into 8869 semantic and 10675 syntactic questions. A question is correctly answered only if the algorithm selects the word that is exactly the same as the correct word in the question: synonyms are thus counted as mistakes. There are two ways to answer these questions, namely, by using 3CosAdd or 3CosMul (see (Levy and Goldberg, 2014) for details). We will report scores by using 3CosAdd by default, and indicate when 3CosMul gives better performance.

4.2 The impact of $r_{w,c}$ and $\rho(\cdot)$

In Sec. 2.2 we argued the need for adding weight $r_{w,c}$ to ranking objective (3), and we also presented our framework which can deal with a variety of ranking loss functions ρ. We now study the utility of these two ideas. We report results on the 17 million token dataset in Table 2. For the similarity task, we use the WS-353 test set and for the analogy task we use the Google analogy test set. The best scores for each task are underlined. We set $t = 1.5$ for ρ_3. "Off" means that we used uniform weight $r_{w,c} = 1$, and "on" means that $r_{w,c}$ was set as in (4). For comparison, we also include the results using RobiRank (Yun et al., 2014)[9].

It can be seen from Table 2 that adding the weight $r_{w,c}$ improves performance in all the cases, especially on the word analogy task. Among the four ρ functions, ρ_0 performs the best on the word similarity task but suffers notably on the analogy task, while $\rho_1 = \log$ performs the best overall. Given these observations, which are consistent with the results on large scale datasets, in the experiments that follow we only report *WordRank* with the best configuration, *i.e.*, using ρ_1 with the weight $r_{w,c}$ as defined in (4).

4.3 Comparison to state-of-the-arts

In this section we compare the performance of *WordRank* with *word2vec*[10] and *GloVe*[11], by using the code provided by the respective authors. For a fair comparison, *GloVe* and *WordRank* are given as input the same co-occurrence matrix X; this eliminates differences in performance due to window size and other such artifacts, and the same parameters are used to *word2vec*. Moreover, the embedding dimensions used for each of the three methods is the same (see Table 1). With *word2vec*, we train the Skip-Gram with Negative Sampling (SGNS) model since it produces state-of-the-art performance, and is widely used in the NLP community (Mikolov et al., 2013b). For *GloVe*, we use the default parameters as suggested by (Pennington et al., 2014). The results are provided in Figure 2 (also see Table 4 in the supplementary material for additional details).

As can be seen, when the size of corpus increases, in general all three algorithms improve their prediction accuracy on both tasks. This is to be expected since a larger corpus typically produces better statistics and less noise in the co-occurrence matrix X. When the corpus size is small (*e.g.*, 17M, 32M, 64M, 128M), *WordRank* yields the best performance with significant margins among three, followed by *word2vec* and *GloVe*; when the size of corpus increases further, on the word analogy task *word2vec* and *GloVe* become very competitive to *WordRank*, and eventually perform neck-to-neck to each other (Figure 2(b)). This is consistent with the findings of (Levy et al., 2015) indicating that when the number of tokens is large even simple algorithms can perform well. On the other hand, *WordRank* is dominant on the word similarity task for all the cases (Figure 2(a)) since it optimizes a ranking loss *explicitly*, which aligns more naturally with the objective of word similarity than the other methods; with 17 million tokens our method performs almost as well as existing methods using 7.2 billion tokens on the word similarity benchmark.

To further evaluate the model performance on the word similarity/analogy tasks, we use the best performing models trained on the 7.2-billion-token corpus to predict on the six word similarity datasets described in Sec. 4.1. Moreover, we breakdown the performance of the models on the Google word analogy dataset into the semantic and syntactic subtasks. Results are listed in Table 3. As can be seen, *WordRank* outperforms *word2vec* and *GloVe* on 5 of 6 similarity tasks, and 1 of 2 Google analogy subtasks.

[9] We used the code provided by the authors at `https://bitbucket.org/d_ijk_stra/robirank`. Although related to RobiRank, we attribute the superior performance of *WordRank* to the use of weight $r_{w,c}$ (4), introduction of hyperparameters α and β, and many implementation details.

[10] `https://code.google.com/p/word2vec/`

[11] `http://nlp.stanford.edu/projects/glove`

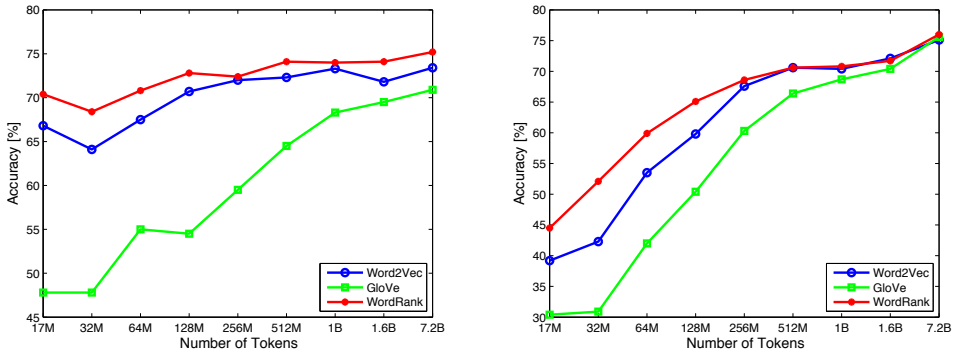

Figure 2: Performance evolution as a function of corpus size (a) on WS-353 word similarity benchmark; (b) on Google word analogy benchmark.

Model	Word Similarity						Word Analogy	
	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. MT	Luong et al. RW	Hill et al. SimLex	Goog Sem.	Goog Syn.
word2vec	73.9	60.9	75.4	66.4	45.5	36.6	78.8	72.0
GloVe	75.7	67.5	78.8	69.7	43.6	41.6	80.9	71.1
WordRank	79.4	70.5	78.1	73.5	47.4	43.5	78.4	74.7

Table 3: Performance of the best *word2vec*, *GloVe* and *WordRank* models, learned from 7.2 billion tokens, on six similarity tasks and Google semantic and syntactic subtasks.

5 Visualizing the results

To understand whether *WordRank* produces *syntatically* and *semantically* meaningful vector space, we did the following experiment: we use the best performing model produced using 7.2 billion tokens, and compute the nearest neighbors of the word "cat". We then visualize the words in two dimensions by using t-SNE (Maaten and Hinton, 2008). As can be seen in Figure 3, our ranking-based model is indeed capable of capturing both semantic (*e.g.*, cat, feline, kitten, tabby) and syntactic (*e.g.*, leash, leashes, leashed) regularities of the English language.

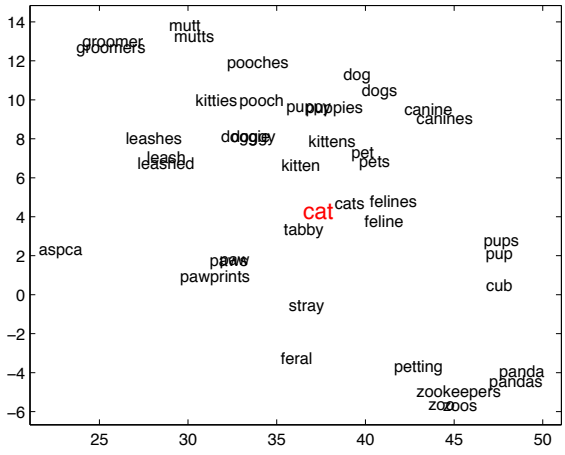

Figure 3: Nearest neighbors of "cat" found by projecting a 300d word embedding learned from *WordRank* onto a 2d space.

6 Conclusion

We proposed *WordRank*, a ranking-based approach, to learn word representations from large scale textual corpora. The most prominent difference between our method and the state-of-the-art techniques, such as *word2vec* and *GloVe*, is that *WordRank* learns word representations via a robust ranking model, while *word2vec* and *GloVe* typically model a transformation of co-occurrence count $X_{w,c}$ directly. Moreover, by a ranking loss function $\rho(\cdot)$, *WordRank* achieves its attention mechanism and robustness to noise naturally, which are usually lacking in other ranking-based approaches. These attributes significantly boost the performance of *WordRank* in the cases where training data are sparse and noisy. Our multi-node distributed implementation of *WordRank* is publicly available for general usage.

Acknowledgments

We'd like to thank Omer Levy for sharing his script for preprocessing the corpora used in the paper. We also thank the anonymous reviewers for their valuable comments and suggestions.

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. *Proceedings of Human Language Technologies*, pages 19–27.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. 2015. Random walks on context spaces: Towards an explanation of the mysteries of semantic word embeddings. Technical report, ArXiV. `http://arxiv.org/pdf/1502.03520.pdf`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe. 2006. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156.

Léon Bottou and Olivier Bousquet. 2011. The tradeoffs of large-scale learning. *Optimization for Machine Learning*, page 351.

Elia Bruni, Gemma Boleda, Marco Baroni, and Nam Khanh Tran. 2012. Distributional semantics in technicolor. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 136–145.

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. ACM.

T. M. Cover and J. A. Thomas. 1991. *Elements of Information Theory*. John Wiley and Sons, New York.

Nan Ding and S. V. N. Vishwanathan. 2010. t-logistic regression. In Richard Zemel, John Shawe-Taylor, John Lafferty, Chris Williams, and Alan Culota, editors, *Advances in Neural Information Processing Systems 23*.

J. J. Dongarra, J. Du Croz, S. Duff, and S. Hammarling. 1990. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16:1–17.

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*, 20:116–131.

R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Conference on Knowledge Discovery and Data Mining*, pages 69–77.

Felix Hill, Roi Reichart, and Anna Korhonen. 2014. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*.

Hugo Larochelle and Geoffrey E. Hinton. 2010. Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS) 23*, pages 1243–1251.

Ching-Pei Lee and Chih-Jen Lin. 2013. Large-scale linear ranksvm. *Neural Computation*. To Appear.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In Max Welling, Zoubin Ghahramani, Corinna Cortes, Neil Lawrence, and Kilian Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2177–2185.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. 2015. Learning semantic word embeddings based on ordinal knowledge constraints. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1501–1511.

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

L. van der Maaten and G.E. Hinton. 2008. Visualizing high-dimensional data using t-sne. *jmlr*, 9:2579–2605.

C. D. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In Chris Burges, Leon Bottou, Max Welling, Zoubin Ghahramani, and Kilian Weinberger, editors, *Advances in Neural Information Processing Systems 26*.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NIPS) 27*, pages 2204–2212.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12.

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

R. T. Rockafellar. 1970. *Convex Analysis*, volume 28 of *Princeton Mathematics Series*. Princeton University Press, Princeton, NJ.

Nicolas Usunier, David Buffoni, and Patrick Gallinari. 2009. Ranking with ordered weighted pairwise classification. In *Proceedings of the International Conference on Machine Learning*.

Luke Vilnis and Andrew McCallum. 2015. Word representations via gaussian embedding. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ling Wang, Chu-Cheng Lin, Yulia Tsvetkov, Silvio Amir, Ramon Fernandez Astudillo, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Not all contexts are created equal: Better word representations with variable attention. In *EMNLP*.

Jason Weston, Chong Wang, Ron Weiss, and Adam Berenzweig. 2012. Latent collaborative retrieval. *arXiv preprint arXiv:1206.4603*.

Hyokun Yun, Parameswaran Raman, and S. V. N. Vishwanathan. 2014. Ranking via robust binary classification and parallel parameter estimation in large-scale data. In *nips*.