On Approximately Searching for Similar Word Embeddings

Kohei Sugawara **Hayato Kobayashi** **Masajiro Iwasaki**
Yahoo Japan Corporation
1-3 Kioicho, Chiyoda-ku, Tokyo 102-8282, Japan
{ksugawar, hakobaya, miwasaki}@yahoo-corp.jp

Abstract

We discuss an approximate similarity search for word embeddings, which is an operation to approximately find embeddings close to a given vector. We compared several metric-based search algorithms with hash-, tree-, and graph-based indexing from different aspects. Our experimental results showed that a graph-based indexing exhibits robust performance and additionally provided useful information, e.g., vector normalization achieves an efficient search with cosine similarity.

1 Introduction

An embedding or distributed representation of a word is a real-valued vector that represents its "meaning" on the basis of *distributional semantics*, where the meaning of a word is determined by its context or surrounding words. For a given meaning space, searching for similar embeddings is one of the most basic operations in natural language processing and can be applied to various applications, e.g., extracting synonyms, inferring the meanings of polysemous words, aligning words in two sentences in different languages, solving analogical reasoning questions, and searching for documents related to a query.

In this paper, we address how to quickly and accurately find similar embeddings in a continuous space for such applications. This is important from a practical standpoint, e.g., when we want to develop a real-time query expansion system on a search engine on the basis of an embedding similarity. A key difference from the existing work is that embeddings are not high-dimensional sparse (traditional count) vectors, but (relatively) low-dimensional dense vectors. We therefore need to use approximate search methods instead of inverted-index-based methods (Zobel and Moffat, 2006). Three types of indexing are generally used in approximate similarity search: hash-, tree-, and graph-based indexing. Hash-based indexing is the most common in natural language processing due to its simplicity, while tree/graph-based indexing is preferred in image processing because of its performance. We compare several algorithms with these three indexing types and clarify which algorithm is most effective for similarity search for word embeddings from different aspects.

To the best of our knowledge, no other study has compared approximate similarity search methods focusing on neural word embeddings. Although one study has compared similarity search methods for (count-based) vectors on the basis of distributional semantics (Gorman and Curran, 2006), our study advances this topic and makes the following contributions: (a) we focus on neural word embeddings learned by a recently developed skip-gram model (Mikolov, 2013), (b) show that a graph-based search method clearly performs better than the best one reported in the Gorman and Curran study from different aspects, and (c) report the useful facts that normalizing vectors can achieve an effective search with cosine similarity, the search performance is more strongly related to a learning model of embeddings than its training data, the distribution shape of embeddings is a key factor relating to the search performance, and the final performance of a target application can be far different from the search performance. We believe that our timely results can lead to the practical use of embeddings, especially for real-time applications in the real world.

The rest of the paper is organized as follows. In Section 2, we briefly survey hash-, tree-, and graph-based indexing methods for achieving similarity search in a metric space. In Section 3, we

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 2265–2275,
Berlin, Germany, August 7-12, 2016.

compare several similarity search algorithms from different aspects and discuss the results. Finally, Section 4 concludes the paper.

2 Similarity Search

We briefly survey similarity search algorithms for real-valued vectors, where we focus on approximate algorithms that can deal with large scale data. In fact, word embeddings are usually trained on a very large corpus. For example, well known pre-trained word embeddings (Mikolov, 2013) were trained on the Google News dataset and consist of about 1,000 billion words with 300-dimensional real-valued vectors. Search tasks on large-scale real-valued vectors have been more actively studied in the image processing field than in the natural language processing field, since such tasks naturally correspond to searching for similar images with their feature vectors.

Many similarity search algorithms have been developed and are classified roughly into three indexing types: hash-, tree-, and graph-based. In natural language processing, hash-based indexing seems to be preferred because of its simplicity and ease of treating both sparse and dense vectors, while in image processing, tree- and graph-based indexing are preferred because of their performance and flexibility in adjusting parameters. We explain these three indexing types in more detail below.

2.1 Hash-based Indexing

Hash-based indexing is a method to reduce the dimensionality of high-dimensional spaces by using some hash functions so that we can efficiently search in the reduced space. Locality-sensitive hashing (LSH) (Gionis et al., 1999) is a widely used hash-based indexing algorithm, which maps similar vectors to the same hash values with high probability by using multiple hash functions.

There are many hash-based indexing algorithms that extend LSH for different metric spaces. Datar et al. (2004) applied the LSH scheme to L^p spaces, or Lebesgue spaces, and experimentally showed that it outperformed the existing methods for the case of $p = 2$. Weiss et al. (2009) showed that the problem of finding the best hash function is closely related to the problem of graph partitioning and proposed an efficient approximate algorithm by reducing the problem to calculating thresholded eigenvectors of the graph Laplacian. In this paper, we focus on approximation of k-nearest neighbors and are not concerned about the hash-based indexing algorithms, since they are basically designed for finding (not k-nearest) neighbors within a fixed radius of a given point, i.e., a so-called *radius search*.

2.2 Tree-based Indexing

Tree-based indexing is used to recursively divide the entire search space into hierarchical subspaces, where the subspaces are not necessarily disjointed, so that the search space forms a tree structure. Given a search query, we can efficiently find the subspaces including the query by descending from the root note to the leaf nodes in the tree structure and then obtain its search results by scanning only neighbors belonging to the subspaces. Note that in contrast to the hash-based indexing, we can easily extend the size of search results or the number of nearest neighbors by ascending to the parent subspaces.

Arya et al. (1998) proposed the balanced box-decomposition tree (BBD-tree) as a variant of the kd-tree (Bentley, 1975) for approximately searching for similar vectors on the basis of Minkowski metrics, i.e., in L^p spaces when $p \geq 1$. Fast library for approximate nearest neighbors (FLANN) (Muja and Lowe, 2008) is an open-source library for approximate similarity search. FLANN automatically determines the optimal one from three indices: a randomized kd-tree where multiple kd-trees are searched in parallel (Silpa-Anan and Hartley, 2008), a k-means tree that is constructed by hierarchical k-means partitioning (Nister and Stewenius, 2006), and a mix of both kd-tree and k-means tree. Spatial approximation sample hierarchy (SASH) (Houle and Sakuma, 2005) achieves approximate search with multiple hierarchical structures created by random sampling. According to the results in the previous study (Gorman and Curran, 2006), SASH performed the best for vectors on the basis of distributional semantics, and its performance surpassed that of LSH.

2.3 Graph-based Indexing

Graph-based indexing is a method to approximately find nearest neighbors by using a neighborhood graph, where each node is connected to its nearest neighbors calculated on the basis of a certain metric. A simple search procedure for a given query is achieved as follows. An arbitrary node in the graph is selected as a candidate for the

true nearest neighbor. In the process of checking the nearest neighbor of the candidate, if the query is closer to the neighbor than the candidate, the candidate is replaced by the neighbor. Otherwise, the search procedure terminates by returning the current candidate as the nearest neighbor of the query. This procedure can be regarded as a best-first search, and the result is an approximation of that of an exact search.

Sebastian and Kimia (2002) first used a k-nearest neighbor graph (KNNG) as a search index, and Hajebi et al. (2011) improved the search performance by performing hill-climbing starting from a randomly sampled node of a KNNG. Their experimental results with image features, i.e., scale invariant feature transform (SIFT), showed that a similarity search based on a KNNG outperforms randomized kd-trees and LSH. Although the brute force construction cost of a KNNG drastically increases as the number of nodes increases because the construction procedure needs to calculate the nearest neighbors for each node, we can efficiently approximate a KNNG (so-called ANNG) by incrementally constructing an ANNG with approximate k-nearest neighbors calculated on a partially constructed ANNG. Neighborhood graph and tree for indexing (NGT) (Iwasaki, 2015) is a library released from Yahoo! JAPAN that achieves a similarity search on an ANNG; it has already been applied to several services.

3 Experiments

In this paper, we focused on the pure similarity search task of word embeddings rather than complex application tasks for avoiding extraneous factors, since many practical tasks can be formulated as k-nearest neighbor search. For example, assuming search engines, we can formalize query expansion, term deletion, and misspelling correction as finding frequent similar words, infrequent similar words, and similar words with different spellings, respectively.

We chose FLANN from the tree-based methods and NGT from the graph-based methods since they are expected to be suitable for practical use. FLANN and NGT are compared with SASH, which was the best method reported in a previous study (Gorman and Curran, 2006). In addition, we consider LSH only for confirmation, since it is widely used in natural language processing, although several studies have reported that LSH performed worse than SASH and FLANN. We used the E2LSH package (Andoni, 2004), which includes an implementation of a practical LSH algorithm.

3.1 Problem Definition

The purpose of an approximate similarity search is to quickly and accurately find vectors close to a given vector. We formulate this task as a problem to find k-nearest neighbors as follows. Let (X, d) be a metric space. We denote by $N_k(x, d)$ the set of k-nearest neighbors of a vector $x \in X$ with respect to a metric d. Formally, the following condition holds: $\forall y \in N_k(x, d), \forall z \in X \setminus N_k(x, d)$, $d(x, y) \leq d(x, z)$. Our goal with this problem is to approximate $N_k(x, d)$ for a given vector x.

We calculate the precision of an approximate search method A using the so-called *precision at k or P@k*, which is a widely used evaluation measure in information retrieval. The precision at k of A is defined as $|N_k(x, d) \cap \tilde{N}_k(x, A)|/k$, where $\tilde{N}_k(x, A)$ is the set of approximate k-nearest neighbors of a vector x calculated by A. Since we use the same size k for an exact set $N_k(x, d)$ and its approximate set $\tilde{N}_k(x, A)$, there is no trade-off between precision and recall.

3.2 Basic Settings

This section describes the basic settings in our experiments, where we changed a specific setting (e.g., number of dimensions) in order to evaluate the performance in each experiment. All the experiments were conducted on machines with two Xeon L5630 2.13-GHz processors and 24 GB of main memory running Linux operating systems.

We prepared 200-dimensional word embeddings learned from English Wikipedia in February 2015, which contains about 3 billion sentences spanning about 2 million words and 35 billion tokens, after preprocessing with the widely used script (Mahoney, 2011), which was also used for the word2vec demo (Mikolov, 2013). We used the skip-gram learning model with hierarchical softmax training in the word2vec tool, where the window size is 5, and the down-sampling parameter is 0.001.

We constructed and evaluated the index by dividing the learned embeddings into 2 million embeddings for training and 1,000 embeddings for testing by random sampling, after normalizing them so that the norm of each embedding was one. We built the search index of each search method

for the training set on the basis of the Euclidean distance. The Euclidean distance of normalized vectors is closely related to the cosine similarity, as described later. We prepared the top-10 (exact) nearest neighbors in the training set corresponding to each embedding in the testing set and plotted the average precision at 10 over the test set versus its computation time (log-scale), by changing the parameter for precision of each method as described below. Note that it is difficult to compare different algorithms in terms of either precision or computation time, since there is a trade-off between precision and computation time in approximate search.

We set the parameters of the three search methods SASH, FLANN, and NGT as follows. We determined stable parameters for indexing using grid search and changed an appropriate parameter that affected the accuracy when evaluating each method. For confirmation, we added LSH in the first experiment but did not use it in the other experiments since it clearly performs worse than the other methods.

SASH We set the maximum number (p) of parents per node to 6 for indexing and changed the scale factor for searching[1].

FLANN We set the target precision to 0.8, the build weight to 0, and the sample fraction to 0.01 for indexing, and we changed the number of features to be checked in the search[2]. The k-means index was always selected as the optimal index in our experiments.

NGT We set the edge number (E) to 10 for indexing and changed the search range (e) for searching.

LSH We set the success probability ($1 - \delta$) to 0.9 and changed the radius (R) for indexing. Note that there are no parameters for searching since LSH was developed to reduce dimensionality, and we need to construct multiple indices for adjusting its accuracy.

[1]The scale factor is implemented as "scaleFactor" in the source code (Houle, 2005), although there is no description in the original paper (Houle and Sakuma, 2005).

[2]Since FLANN is a library integrating several algorithms, the parameters can be described only by variables in the source code (Muja and Lowe, 2008). The target precision, build weight, and sample fraction for auto-tuned indexing are implemented as "target_precision", "build_weight", and "sample_fraction" in the structure "AutotunedIndexParams", respectively. The number of features is implemented as "checks" in the structure "SearchParams".

3.3 Results

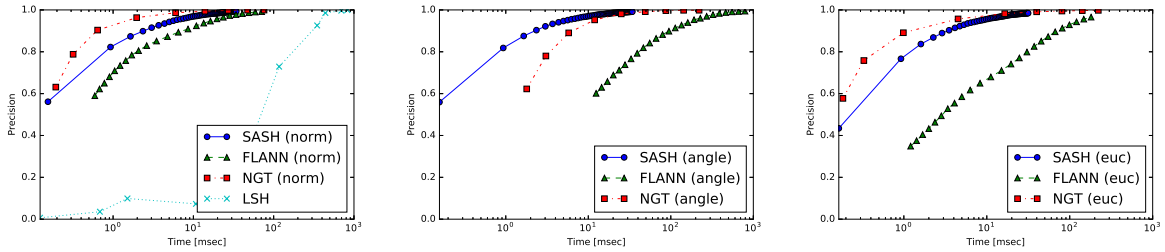
In this section we report the results of the performance comparison of SASH, FLANN, and NGT from the following different aspects: the distance function for indexing, the number of dimensions of embeddings, the number of neighbors to be evaluated, the size of a training set for indexing, the learning model/data used for embeddings, and the target task to be solved.

3.3.1 Distance Function for Indexing

We evaluated the performance by changing the distance function for indexing. In natural language processing, cosine similarity $\cos(x, y) = \frac{x \cdot y}{\|x\| \, \|y\|}$ of two vectors x and y is widely used from a practical perspective, and cosine distance $d^{\cos}(x, y) = 1 - \cos(x, y)$ as its complement seems to be appropriate for the distance function for indexing. Unfortunately, however, the cosine distance is not strictly metric but *semimetric* since the triangle inequality is not satisfied. Thus, we cannot directly use the cosine distance because the triangle inequality is a key element for efficient indexing in a metric space. In this paper, we use two alternatives: normalized and angular distances.

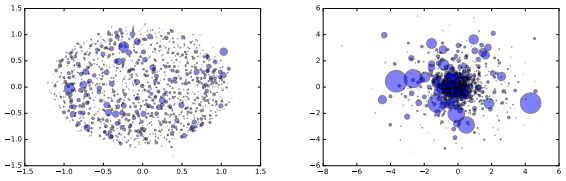
The former is the Euclidean distance after normalizing vectors, i.e., $d^{\text{norm}}(x, y) = d^{\text{euc}}(\frac{x}{\|x\|}, \frac{y}{\|y\|})$, where $d^{\text{euc}}(x, y) = \|x - y\|$. The set of k-nearest neighbors by d^{norm} is theoretically the same as that by $d^{\cos}$, i.e., $N_k(x, d^{\text{norm}}) = N_k(x, d^{\cos})$, since $d^{\text{norm}}(x, y)^2 = \frac{\|x\|^2}{\|x\|^2} + \frac{\|y\|^2}{\|y\|^2} - 2\frac{x}{\|x\|} \cdot \frac{y}{\|y\|} = 2d^{\cos}(x, y)$. The latter is the angle between two vectors, i.e., $d^{\text{arc}}(x, y) = \arccos(\cos(x, y))$. The set of k-nearest neighbors by d^{arc} is also the same as that by $d^{\cos}$, i.e., $N_k(x, d^{\text{arc}}) = N_k(x, d^{\cos})$, since $\arccos$ is a monotone decreasing function. Note that d^{arc} is not strictly metric, but it satisfies the triangle inequality, i.e., *pseudometric*.

Figure 1 plots the performances of SASH, FLANN, and NGT using the normalized, angular, and ordinal Euclidean distances. Higher precision at the same computational time (upper left line) indicates a better result. The graphs show that NGT performed the best for the normalized distance (a), while SASH performed the best for the angular distance (b). This large difference is caused by the long computational time of d^{arc}. Because we only want the maximum performance in graphs (a) and (b) for each method, we used only the normalized distance in the later experiments since the perfor-



(a) Normalized (b) Angular (c) Euclidean

Figure 1: Precision versus computation time of SASH, FLANN, and NGT using the normalized, angular, and Euclidean distances.



(a) Normalized (b) Un-normalized

Figure 2: 2D visualization of normalized and un-normalized embeddings by multi-dimensional scaling.

Distance	Method	Time (min)
Normalized	SASH	74.6
	FLANN	56.5
	NGT	**33.9**
	LSH	44.6
Angular	SASH	252.4
	FLANN	654.9
	NGT	**155.4**
Euclidean	SASH	58.1
	FLANN	**20.2**
	NGT	83.0

Table 1: Indexing time of SASH, FLANN, NGT, and LSH using the normalized, angular, Euclidean distance functions.

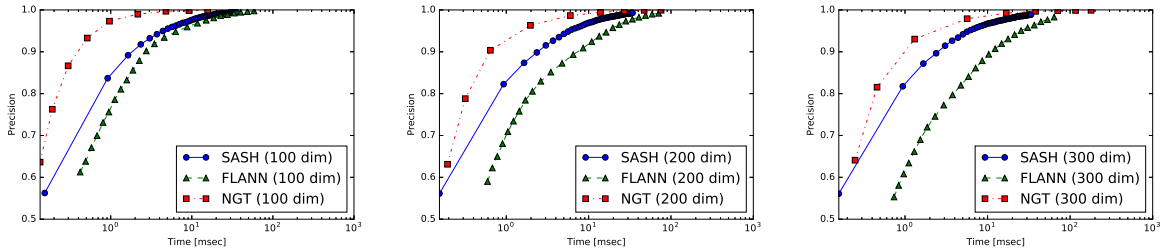
mance of SASH in graph (a) is almost the same as that in (b). For confirmation, we added the result of LSH in graph (a) only. The graph clearly indicates that the performance of LSH is very low even for neural word embeddings, which supports the results in the previous study (Gorman and Curran, 2006), and therefore we did not use LSH in the later experiments.

Graph (c) shows that the performance using the Euclidean distance has a similar tendency to that using the normalized distance, but its computation time is much worse than that using the normalized distance. The reason for this is that it is essentially difficult to search for distant vectors in a metric-based index, and normalization can reduce the number of distant embeddings by aligning them on a hypersphere. In fact, we can confirm that the number of distant embeddings was reduced after normalization according to Figure 2, which visualizes 1,000 embeddings before/after normalization on a two-dimensional space by multi-dimensional scaling (MDS) (Borg and Groenen, 2005), where the radius of each circle represents the search time of the corresponding embedding calculated by NGT. MDS is a dimensionality reduction method to place each point in a low-dimensional space such that the distances between any two points are preserved as much as possible. Note that the scale of graph (b) is about five times larger than that of graph (a). This also suggests that the normalized distance should be preferred even when it has almost the same precision as the Euclidean distance.

Table 1 lists the indexing times of SASH, FLANN, and NGT on the basis of the normalized, angular, and Euclidean distances, where LSH is also added only in the result of the normalized distance. The table indicates that NGT performed the best for the normalized and angular distances, while FLANN performed the best for the Euclidean distance. However, all methods seem to be suitable for practical use in terms of indexing because we can create an index of English Wikipedia embeddings in several hours (only once). The large indexing time with the angular distance also supports our suggestion that the normalized distance should be used.

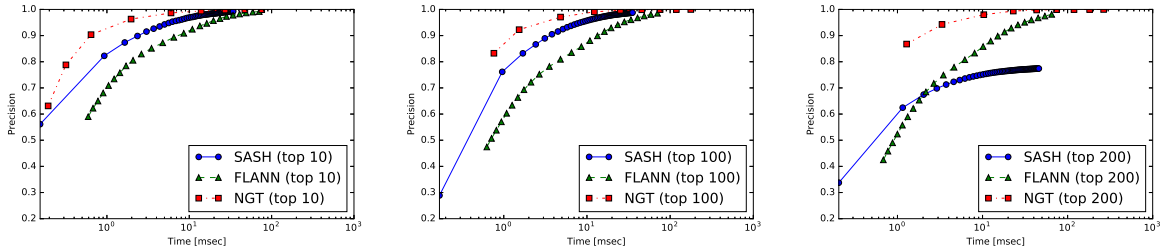
3.3.2 Number of Dimensions of Embeddings

We also evaluated the performances by changing the number of dimensions of embeddings. Since the optimal number of dimensions should depend on the tasks, we wanted to see how the search



(a) 100 dimensions (b) 200 dimensions (c) 300 dimensions

Figure 3: Precision versus computation time of SASH, FLANN, and NGT using 100-, 200-, and 300-dimensional embeddings.



(a) P@10 (b) P@100 (c) P@200

Figure 4: Precision versus computation time of SASH, FLANN, and NGT using precision at 10, 100, and 200.

methods performed when the number of dimensions varied, while the number of dimensions of image features is usually fixed. For example, SIFT features (Lowe, 1999) are represented as 128-dimensional vectors.

Figure 3 plots the performances of SASH, FLANN, and NGT using 100-, 200-, and 300-dimensional embeddings. The graphs indicate that NGT always performed the best. SASH is expected to perform well when the number of dimensions is large, since FLANN and NGT perform worse as the number of dimensions becomes larger. However, NGT would be a better choice since most existing pre-trained embeddings (Turian et al., 2010; Mikolov, 2013; Pennington et al., 2014a) have a few hundred dimensions.

3.3.3 Number of Neighbors to Be Evaluated

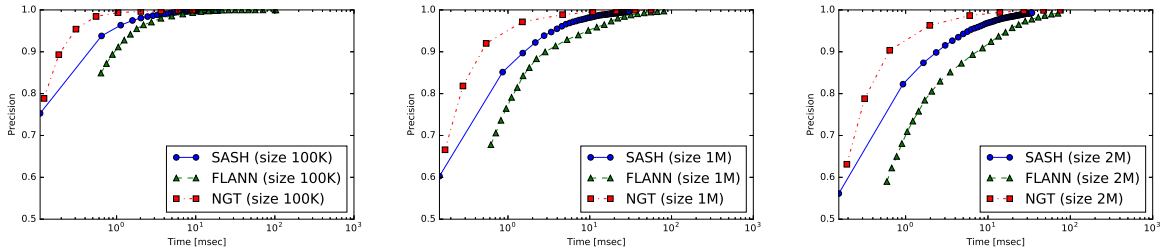
We also conducted performance evaluations by changing the number k of neighbors, i.e., the size of the set of k-nearest neighbors, to calculate the precision at k. We need to change the number k on demand from target applications. For example, we may use small numbers for extracting synonyms and large numbers for selecting candidates for news recommendations, where they will be reduced via another sophisticated selection process.

The performances of SASH, FLANN, and NGT using 10-, 100-, and 200-nearest neighbors are shown in Figure 4. The graphs indicate that NGT performed the best in this measure also. With 200-nearest neighbors, the performance of SASH dropped sharply, which means that SASH is not robust for the indexing parameter. One possible reason is that searching for relatively distant neighbors is difficult for a tree-based index, where the divided subspaces are not appropriate.

3.3.4 Size of Training Set for Indexing

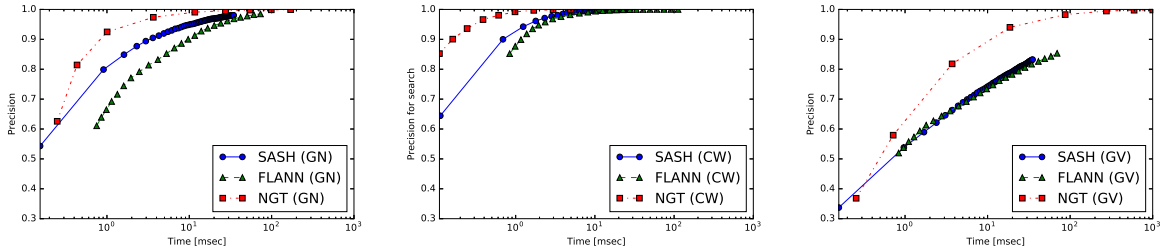
We conducted further performance evaluations by changing the size of a training set, i.e., the number of embeddings used for indexing. We wanted to know how the search methods performed with different sized search indices since a large search index will bring about extra operational costs in a practical sense, and a small search index is preferred for a small application system.

Figure 5 plots the performances of SASH, FLANN, and NGT using 100K, 1M, and 2M training sets, which were randomly sampled so that each training set can be virtually regarded as embeddings with a vocabulary of its training set size. The graphs indicate that NGT always performed the best for all search index sizes. Moreover, we can see that all results for each method have a similar tendency. This fact implies that a distribution of embeddings is related to the search per-



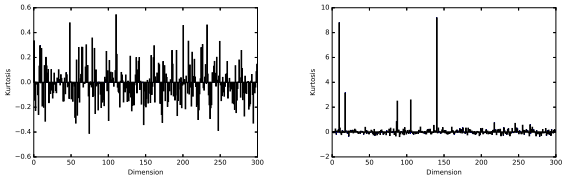
(a) 100K (b) 1M (c) 2M

Figure 5: Precision versus computation time of SASH, FLANN, and NGT using 100K, 1M, and 2M training sets.



(a) GN (b) CW (c) GV

Figure 6: Precision versus computation time of SASH, FLANN, and NGT using GN, CW, and GV embeddings.



(a) GN (b) GV

Figure 7: Kurtosis of each dimension of GN and GV embeddings.

formance, and the next section will actually confirm the same property on another dataset used for learning embeddings.

3.3.5 Model and Data Used for Embeddings

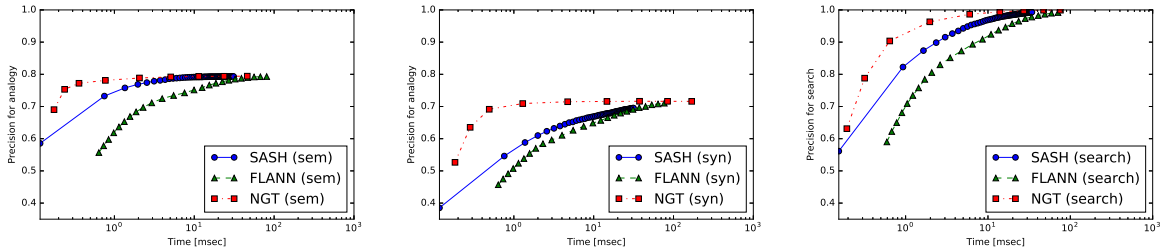
We also conducted performance evaluations by changing the learning models and training data for embeddings. We used the following three pre-trained embeddings to investigate the performance when changing the data distributions used for indexing.

GN 300-dimensional embeddings (Mikolov, 2013) learned by the skip-gram model with negative sampling (Mikolov et al., 2013a) using part of the Google News dataset, which contains about 3 million words and phrases and 100 billion tokens.

CW 200-dimensional embeddings (Turian et al., 2010) learned by deep neural networks (Collobert and Weston, 2008) using the RCV1 corpus, which contains about 269 thousand words and 63 million tokens.

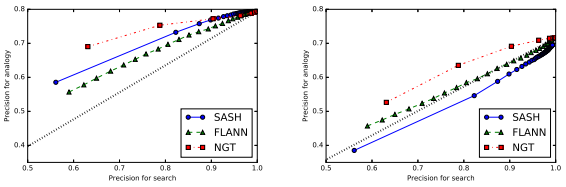
GV 300-dimensional embeddings (Pennington et al., 2014a) learned by the global vectors for word representation (GloVe) model (Pennington et al., 2014b) using Common Crawl corpora, which contain about 2 million words and 42 billion tokens.

The performances of SASH, FLANN, and NGT using GN, CW, and GV embeddings are plotted in Figure 6. The graphs indicate that NGT consistently performed the best over different learning models. A comparison of the results using GN embeddings and the previous results using Wikipedia embeddings reveals that they had almost the same tendency. This fast can be acceptable assuming an empirical rule that a corpus follows a power law or Zipf's law. On the other hand, graphs (a), (b), and (c) have quite different tendencies. Specifically, all search methods compete with each other for CW embeddings, while they could not perform well for GV embeddings. This implies that the performance of a search method can be affected by learning models rather than training sets used for embeddings.



(a) Semantic analogy (b) Syntactic analogy (c) Similarity search

Figure 8: Precision versus computation time of SASH, FLANN, and NGT using the semantic analogy, syntactic analogy, and similarity search tasks.



(a) Semantic analogy (b) Syntactic analogy

Figure 9: Precision of the semantic and syntactic analogy tasks versus that of the similarity search task.

	EW	GN	CW	GV
Variance	0.0033	0.0033	0.0050	0.0033
Kurtosis	0.034	-0.026	-0.075	0.57

Table 2: Variance and kurtosis of English Wikipedia (EW), GN, CW, and GV embeddings.

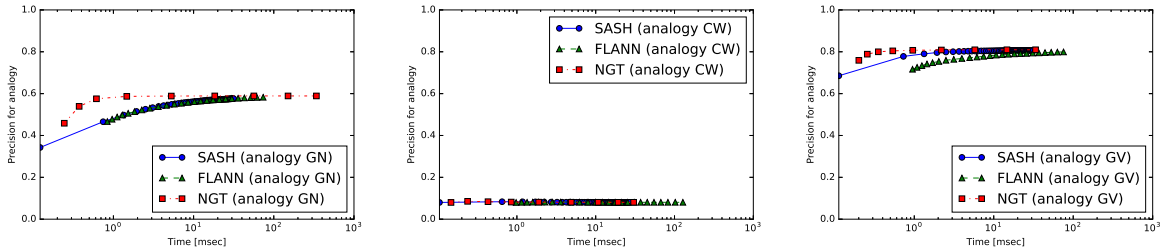
We further investigated why GV embeddings deteriorate the search performance. Table 2 lists the variance and kurtosis of Wikipedia, GN, CW, and GV embeddings for clarifying the variation or dispersion of these distributions. Kurtosis $K(X)$ is a measure of the "tailedness" of the probability distribution of a random variable X, defined by $K(X) = \mu_4/\mu_2^2 - 3$, where μ_n represents the n-th central moment, i.e., $\mathbb{E}[(X - \mathbb{E}[X])^n]$. The constant "3" in the above definition sets the kurtosis of a normal distribution to 0. The table clearly indicates that GV has a heavy tailed distribution in accordance with the kurtosis values, although all variances have almost the same value. In fact, GV has several high kurtosis peaks, while GN has only small values, according to Figure 7, which visualizes the kurtosis of each dimension. Note that the y-axis scale of graph (b) is about 20 times larger than that of graph (a). Because distant points in a metric space tend to deteriorate the performance in a search process, we need to pay attention to the distribution shape of embeddings as well as their quality, so as to efficiently search for similar embeddings.

3.3.6 Target Task to Be Solved

We finally evaluated the performance by changing the target task to be solved by using embeddings. We wanted to know how the search methods performed with different task settings since even if the precision of the search task is not good, it might be sufficient for another task to be solved on the basis of similarity search. In this section, we address well known analogy tasks (Mikolov et al., 2013a), where semantic and syntactic analogy questions are considered, e.g., "Which word corresponds to Japan when Paris corresponds to France?", the answer being "Tokyo". These questions can be solved by searching for the nearest neighbors of analogical vectors generated via arithmetic operations., i.e., vec("Paris") − vec("France") + vec("Japan"), where vec(w) represents an embedding of word w.

Figure 8 plots the performances of SASH, FLANN, and NGT using the semantic and syntactic analogy tasks as well as that using the similarity search task (in Figure 1), which is added for comparison. The graphs indicate that NGT clearly performed the best even in the analogy tasks. Comparing the curves of NGT, we can see that those in graphs (a) and (b) are quite different from that in (c), and the analogy precisions can maintain their quality, even when the search precision is about 0.9.

For further analysis, we aligned the precisions of the search task with those of the analogy tasks in Figure 9, where each point represents the results calculated with the same parameters. The dotted line without markers in each graph is a line from the origin (0, 0) to the point where the analogy precision is maximum when the search precision



(a) Analogy by GN (b) Analogy by CW (c) Analogy by GV

Figure 10: Precision versus computation time of SASH, FLANN, and NGT for the analogy task (including both semantic and syntactic questions) using GN, CW, and GV embeddings.

is 1.0, and thus it naively estimates a deterioration rate of the analogy precision on the basis of the search precision. The graphs indicate that the search precision can be far different from the estimated precision of another task. In fact, when the search precision by NGT is 0.8 in Figure 9 (a), the analogy precision 0.75 is unexpectedly high, since the naive estimation is 0.64 calculated by the maximum analogy precision 0.8 times the search precision 0.8. This suggests that it is a good idea to check the final performance of a target application, although the search performance is valuable from a standpoint of general versatility.

Finally, we conducted performance evaluations for the analogy task instead of the search task by changing the learning models and training data for embeddings as in Section 3.3.5, in order to support the robustness of NGT even for an operation more sophisticated than just finding similar words. Figure 10 plots the performances of SASH, FLANN, and NGT for the analogy task including both semantic and syntactic questions using GN, CW, and GV embeddings. The graphs indicate that NGT performed the best over different learning models even for the analogy task. Although the precisions of CW embeddings in graph (b) are very low, the result seems to be acceptable according to the previous work (Mikolov et al., 2013b), which reported that the precisions of a syntactic analogy task using CW embeddings in similar settings were at most 5 % (0.05). The results of GN and GV embeddings in graphs (a) and (c) show a similar tendency to those of Wikipedia embeddings in Figure 8. However, the overall performance for the analogy task using GV embeddings is unexpectedly high, contrary to the results for the search task in Figure 6 (c). One of the reasons is that arithmetic operations for solving analogy questions can reduce kurtosis peaks, although we omitted the kurtosis results due to space limitation. This fact also supports our finding that distant points in a metric space tend to deteriorate the performance in a search process.

4 Conclusion

We investigated approximate similarity search for word embeddings. We compared three methods: a graph-based method (NGT), a tree-based method (FLANN), the SASH method, which was reported to have the best performance in a previous study (Gorman and Curran, 2006). The results of experiments we conducted from various aspects indicated that NGT generally performed the best and that the distribution shape of embeddings is a key factor relating to the search performance. Our future research includes improving the search performance for embeddings with heavy-tailed distributions and creating embeddings that can keep both task quality and search performance high.

We will release the source code used for our comparative experiments from the NGT page (Iwasaki, 2015). Since we need to implement additional glue codes for running FLANN and SASH, our code would be useful for researchers who want to compare their results with ours.

Acknowledgments

We would like to thank the anonymous reviewers for giving us helpful comments.

References

Alexandr Andoni. 2004. LSH Algorithm and Implementation (E2LSH). `http://web.mit.edu/andoni/www/LSH/`.

Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. 1998. An Optimal Algorithm for Approximate Nearest Neighbor

Searching Fixed Dimensions. *Journal of the ACM (JACM)*, 45(6):891–923.

Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communication of the ACM*, 18(9):509–517.

Ingwer Borg and Patrick J. F. Groenen. 2005. *Modern Multidimensional Scaling*. Springer Series in Statistics. Springer-Verlag New York.

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 (ICML 2008)*, pages 160–167. ACM.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry (SCG 2004)*, pages 253–262. ACM.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB 2009)*, pages 518–529. Morgan Kaufmann Publishers Inc.

James Gorman and James R. Curran. 2006. Scaling Distributional Similarity to Large Corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 2006)*, pages 361–368. Association for Computational Linguistics.

Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast Approximate Nearest-neighbor Search with K-nearest Neighbor Graph. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 1312–1317. AAAI Press.

Michael E. Houle and Jun Sakuma. 2005. Fast Approximate Similarity Search in Extremely High-Dimensional Data Sets. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, pages 619–630. IEEE Computer Society.

Michael E. Houle. 2005. The SASH Page. `http://research.nii.ac.jp/%7Emeh/sash/sashpage.html`.

Masajiro Iwasaki. 2015. NGT : Neighborhood Graph and Tree for Indexing. `http://research-lab.yahoo.co.jp/software/ngt/`.

David G. Lowe. 1999. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV 1999)*, pages 1150–1157. IEEE Computer Society.

Matt Mahoney. 2011. About the Test Data. `http://mattmahoney.net/dc/textdata.html`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013a. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 3111–3119. Curran Associates, Inc.

Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. 2013b. 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 (NAACL-HLT 2013)*, pages 746–751. Association for Computational Linguistics.

Tomas Mikolov. 2013. word2vec: Tool for computing continuous distributed representations of words. `https://code.google.com/p/word2vec/`.

Marius Muja and David G. Lowe. 2008. FLANN — Fast Library for Approximate Nearest Neighbors. `http://www.cs.ubc.ca/research/flann/`.

David Nister and Henrik Stewenius. 2006. Scalable Recognition with a Vocabulary Tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006)*, pages 2161–2168. IEEE Computer Society.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014a. GloVe: Global Vectors for Word Representation. `http://nlp.stanford.edu/projects/glove/`.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014b. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543.

Thomas B. Sebastian and Benjamin B. Kimia. 2002. Metric-Based Shape Retrieval in Large Databases. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR 2002)*, pages 291–296.

Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, pages 1–8. IEEE Computer Society.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. CCG: RTE Annotation Data for ACL 2010 publication. `http://cogcomp.cs.illinois.edu/Data/ACL2010_NER_Experiments.php`.

Yair Weiss, Antonio Torralba, and Robert Fergus. 2009. Spectral Hashing. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1753–1760. Curran Associates, Inc.

Justin Zobel and Alistair Moffat. 2006. Inverted Files for Text Search Engines. *ACM Computing Surveys*, 38(2).