Fast Approximate Nearest Neighbor Search*

Amit Goyal and Hal Daumé III
Dept. of Computer Science, University of Maryland, College Park, MD 20742
{amit,hal}@umiacs.umd.edu

Many problems in Computer Vision [6, 5] and Natural Language Processing [12, 3] involves finding l nearest neighbors to the query. However, finding exact l nearest neighbors to the query can be time and memory intensive [10, 9]. Hence, in some applications [10, 5] it may be acceptable to return approximate l nearest neighbors. In this work, we propose a *novel* fast approximate nearest neighbor search algorithm and apply it to finding l similar words with respect to a query word.

1 Preprocessing for Fast Approximate Nearest Neighbor Search

First, for every word "z", we assume that we are given a context vector ($\langle(c_1, v_1); (c_2, v_2) \ldots; (c_d, v_d)\rangle$) of size d where c_d denotes the context and v_d denotes the Pointwise Mutual Information (PMI) (strength of association) between the context c_d and the word "z" (vocabulary of Z words). The context can be lexical, predicate argument structure and dependency units that co-occur with the word "z". For each word, we use hashing to project the context vectors onto k directions. We use k pairwise independent hash functions that maps each of the d context (c_d) dimensions onto $\beta_{d,k} \in \{-1, +1\}$; and compute inner product between $\beta_{d,k}$ and v_d. Next, $\forall k, \sum_d \beta_{d,k}.v_d$ returns the k random projections for each word "z". We store the k random projections for all words (mapped to integers) as a matrix A of size of $k \times Z$.

The mechanism described above generates random projections by implicitly creating a random projection matrix from a set of $\{-1, +1\}$. This idea of creating implicit random projection matrix is motivated by the work on stable random projections [7] and online Locality Sensitive Hash [11]. The idea of generating random projections from the set $\{-1, +1\}$ was originally proposed by [1], then extended by [8].

For fast approximate search, we propose a *novel* approach, which involves two pre-processing steps:

First pre-processing step of fast approximate search is to create a binary matrix B using matrix A by taking sign of each of the entries of the matrix A. If $A(i, j) \geq 0$, then $B(i, j) = 1$; else $B(i, j) = 0$. This binarization creates Locality Sensitive Hash (LSH) function that preserves the cosine similarity between every pair of word vectors. This idea was first proposed by Charikar [2] and used in NLP for large-scale noun clustering [10]. However, in large-scale noun clustering work, they had to store the random projection matrix of size $D \times k$; where D denotes the number of all unique contexts (which is generally large and $D >> Z$) and in this paper, we do not explicitly require to store a random projection matrix.

We then pre-process the matrix A. First for matrix A, we pair the words $1 \cdots Z$ and their random projection values as shown in first matrix in Fig. 1. Second, we sort the elements of each row of matrix A by their random projection values from smallest to largest (shown in second matrix in Fig. 1). The sorting step takes $O(ZlogZ)$ time (We can assume k to be a constant). The sorting operation puts all the nearest neighbor words (for each k independent projections) next to each other. After sorting the matrix A, we throw away the projection values leaving only the words (third matrix in Fig. 1). To search a word in matrix A in constant time, we create another matrix C of size ($k \times Z$) that is the fourth matrix from Fig. 1. Matrix C maps the words $1 \cdots Z$ to their sorted position in the matrix A (third matrix from Fig. 1) for each k.

2 Fast Approximate Search

After the pre-processing is done, fast approximate search is very simple and fast. To search a word "z", first, we can look up matrix C to locate the k positions where "z" is stored in matrix A. This can be done in constant time (Again assuming k to be a constant.). Once, we find "z" in each row, we can select b (beam parameter) neighbors ($b/2$ neighbors from left and $b/2$ neighbors from right of the query word.) for all the k rows. This can be done in

*Topic: other applications and Preference: oral

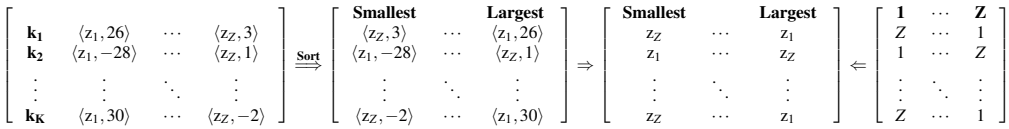

Figure 1: First matrix pairs the words $1 \cdots Z$ and their random projection values. Second matrix sorts each row by the random projection values from smallest to largest. Third matrix throws away the projection values leaving only the words. Fourth matrix maps the words $1 \cdots Z$ to their sorted position in the third matrix for each k. This allows constant query time for all the words.

jazz	yale	soccer	physics	wednesday	brazil	american	bread	man
reggae	harvard	basketball	chemistry	tuesday	ecuador	british	cornbread	woman
rockabilly	cornell	hockey	biology	thursday	nicaragua	canadian	muffins	person
rock	fordham	lacrosse	biochemistry	monday	bolivia	european	waffles	gentleman
indie	tufts	handball	microbiology	friday	guatemala	lithuanian	oatmeal	neighbour
folk	dartmouth	volleyball	science	sunday	argentina	german	breads	boy
ragtime	nyu	badminton	economics	yesterday	panama	korean	baguette	englishman
funk	ucla	softball	psychology	october	cameroon	hungarian	cake	foreigner
banjo	princeton	football	neuroscience	week	mongolia	armenian	pastries	texan
blues	stanford	tennis	oceanography	thurs	colombia	ethiopian	applesauce	bartender
symphonic	loyola	netball	zoology	september	basf	bulgarian	toppings	policeman

Table 1: Sample Top 10 similarity lists returned by fast approximate nearest neighbor search algorithm with $k = 3000$ and $b = 40$.

constant time (Assuming k and b to be constants.). This search procedure produces a set of bk potential nearest neighbors for a query word "z". Next, we compute Hamming distance between query word "z" and the set of potential nearest neighbors from matrix B to return l closest nearest neighbors.

Our fast approximate search procedure is different from the one used in large-scale noun clustering work [10]. In their work, they used the search algorithm PLEB (Point Location in Equal Balls) first proposed by Indyk and Motwani [4] and further improved by Charikar [2]. The improved PLEB algorithm involves generating p random permutations of the binary matrix B, and Ravichandran et al. [10] used $p = 1000$ random permutations in their work, which means storing $p = 1000$ copies of matrix B. In our work, we only use one copy of B and along with that we store A and C to find potential nearest neighbors.

To evaluate the quality of our approximate search algorithm, we fix parameters $k = 3000$ and $b = 40$. We are given a context vectors for $106,733$ words of size $d = 1000$ and using our approximate algorithm, we find top 10 neighbors for each word. Table 1 shows the top 10 most similar words for some words found by algorithm.

References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

[2] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *In Proc. of 34th STOC*, pages 380–388. ACM, 2002.

[3] Dipanjan Das and Slav Petrov. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 600–609, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[4] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613. ACM, 1998.

[5] Prateek Jain, Brian Kulis, and Kristen Grauman. Fast image search for learned metrics. *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, (June):1–8, 2008.

[6] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2:1–19, February 2006.

[7] Ping Li, Kenneth Ward Church, and Trevor Hastie. One sketch for all: Theory and application of conditional random sampling. In *NIPS*, pages 953–960, 2008.

[8] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 287–296. ACM, 2006.

[9] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of EMNLP*, 2009.

[10] Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of ACL*, 2005.

[11] Benjamin Van Durme and Ashwin Lall. Online generation of locality sensitive hash signatures. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 231–235, July 2010.

[12] Leonid Velikovich, Sasha Blair-Goldensohn, Kerry Hannan, and Ryan McDonald. The viability of web-derived polarity lexicons. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 777–785, June 2010.