Sketch Techniques for Scaling Distributional Similarity to the Web

Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian
School of Computing
University of Utah
Salt Lake City, UT 84112
{amitg,jags,hal,suresh}@cs.utah.edu

Abstract

In this paper, we propose a memory, space, and time efficient framework to scale distributional similarity to the web. We exploit sketch techniques, especially the Count-Min sketch, which approximates the frequency of an item in the corpus without explicitly storing the item itself. These methods use hashing to deal with massive amounts of the streaming text. We store all item counts computed from 90 GB of web data in just 2 billion counters (8 GB main memory) of CM sketch. Our method returns semantic similarity between word pairs in O(K) time and can compute similarity between any word pairs that are stored in the sketch. In our experiments, we show that our framework is as effective as using the exact counts.

1 Introduction

In many NLP problems, researchers (Brants et al., 2007; Turney, 2008) have shown that having large amounts of data is beneficial. It has also been shown that (Agirre et al., 2009; Pantel et al., 2009; Ravichandran et al., 2005) having large amounts of data helps capturing the semantic similarity between pairs of words. However, computing distributional similarity (Sec. 2.1) between word pairs from large text collections is a computationally expensive task. In this work, we consider scaling distributional similarity methods for computing semantic similarity between words to Web-scale.

The major difficulty in computing pairwise similarities stems from the rapid increase in the number of unique word-context pairs with the size of text corpus (number of tokens). Fig. 1 shows that

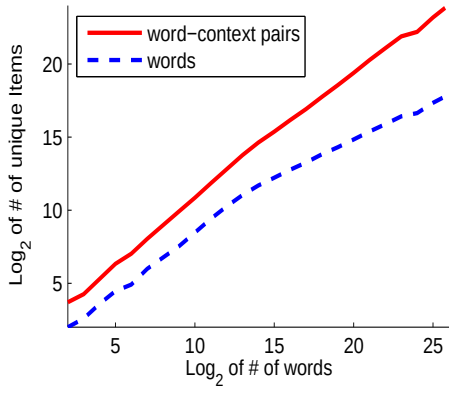

Figure 1: Token Type Curve

the number of unique word-context pairs increase rapidly compared to the number words when plotted against the number of tokens[1]. For example, a 57 million word corpus[2] generates 224 thousand unique words and 15 million unique word-context pairs. As a result, it is computationally hard to compute counts of all word-context pairs with a giant corpora using conventional machines (say with main memory of 8 GB). To overcome this, Agirre et al. (2009) used MapReduce infrastructure (with 2,000 cores) to compute pairwise similarities of words on a corpus of roughly 1.6 Terawords.

In a different direction, our earlier work (Goyal et al., 2010) developed techniques to make the computations feasible on a conventional machines by willing to accept some error in the counts. Similar to that work, this work exploits the idea of Count-Min (CM) sketch (Cormode and Muthukrishnan, 2004) to approximate the frequency of word pairs in the corpus without explicitly storing the word pairs themselves. In their, we stored

[1] Note that the plot is in log-log scale.
[2] 'Subset' column of Table 1 in Section 5.1

counts of all words/word pairs in fixed amount of main memory. We used conservative update with CM sketch (referred as CU sketch) and showed that it reduces the average relative error of its approximate counts by a factor of two. The approximate counts returned by CU Sketch were used to compute approximate PMI between word pairs. We found their that the approximate PMI values are as useful as the exact PMI values for computing semantic orientation (Turney and Littman, 2002) of words. In addition, our intrinsic evaluations in their showed that the quality of approximate counts and approximate PMI is good.

In this work, we use CU-sketch to store counts of items (words, contexts, and word-context pairs) using fixed amount of memory of 8 GB by using only $2B$ *counters*. These approximate counts returned by CU Sketch are converted into approximate PMI between word-context pairs. The top K contexts (based on PMI score) for each word are used to construct distributional profile (DP) for each word. The similarity between a pair of words is computed based on the cosine similarity of their respective DPs.

The above framework of using CU sketch to compute semantic similarity between words has five good properties. First, this framework can return semantic similarity between any word pairs that are stored in the CU sketch. Second, it can return the similarity between word pairs in time O(K). Third, because we do not store items explicitly, the overall space required is significantly smaller. Fourth, the additive property of CU sketch (Sec. 3.2) enables us to parallelize most of the steps in the algorithm. Thus it can be easily extended to very large amounts of text data. Fifth, this easily generalizes to any kind of association measure and semantic similarity measure.

2 Background

2.1 Distributional Similarity

Distributional Similarity is based on the distributional hypothesis (Firth, 1968; Harris, 1985) that words occur in similar contexts tend to be similar. The context of a word is represented by the distributional profile (DP), which contains the strength of association between the word and each of the lexical, syntactic, semantic, and/or dependency units that co-occur with it[3]. The association is commonly measured using conditional probability, pointwise mutual information (PMI) or log likelihood ratios. Then the semantic similarity between two words, given their DPs, is calculated using similarity measures such as Cosine, α-skew divergence, and Jensen-Shannon divergence. In our work, we use PMI as association measure and cosine similarity to compute pairwise similarities.

2.2 Large Scale NLP problems

Pantel et al. (2009) computed similarity between 500 million word pairs using the MapReduce framework from a 200 billion word corpus using 200 quad-core nodes. The inaccessibility of clusters for every one has attracted NLP community to use streaming, and randomized algorithms to handle large amounts of data.

Ravichandran et al. (2005) used locality sensitive hash functions for computing word-pair similarities from large text collections. Their approach stores a enormous matrix of all unique words and their contexts in main memory which makes it hard for larger data sets. In our work, we store all unique word-context pairs in CU sketch with a pre-defined size[4].

Recently, the streaming algorithm paradigm has been used to provide memory and time-efficient platform to deal with terabytes of data. For example, we (Goyal et al., 2009); Levenberg and Osborne (2009) build approximate language models and show their effectiveness in SMT. In (Van Durme and Lall, 2009b), a TOMB Counter (Van Durme and Lall, 2009a) was used to find the top-K verbs "y" with the highest PMI for a given verb "x". The idea of TOMB is similar to CU Sketch. However, we use CU Sketch because of its simplicity and attractive properties (see Sec. 3). In this work, we go one step further, and compute semantic similarity between word-pairs using approximate PMI scores from CU sketch.

2.3 Sketch Techniques

Sketch techniques use a sketch vector as a data structure to store the streaming data compactly in a small-memory footprint. These techniques use hashing to map items in the streaming data onto a small sketch vector that can be easily updated and queried. These techniques generally process the input stream in one direction, say from left to right,

[3] In this work, we only consider lexical units as context.

[4] We use only 2 billion counters which takes up to 8 GB of main memory.

without re-processing previous input. The main advantage of using these techniques is that they require a storage which is significantly smaller than the input stream length. A survey by (Rusu and Dobra, 2007; Cormode and Hadjieleftheriou, 2008) comprehensively reviews the literature.

3 Count-Min Sketch

The Count-Min Sketch (Cormode and Muthukrishnan, 2004) is a compact summary data structure used to store the frequencies of all items in the input stream.

Given an input stream of items of length N and user chosen parameters δ and ϵ, the algorithm stores the frequencies of all the items with the following guarantees:

- All reported frequencies are within ϵN of true frequencies with probability of atleast δ.
- Space used by the algorithm is $O(\frac{1}{\epsilon}\log\frac{1}{\delta})$.
- Constant time of O($\log(\frac{1}{\delta})$) per each update and query operation.

3.1 CM Data Structure

A Count-Min Sketch with parameters (ϵ,δ) is represented by a two-dimensional array with width w and depth d :

$$\begin{bmatrix} \text{sketch}[1,1] & \cdots & \text{sketch}[1,w] \\ \vdots & \ddots & \vdots \\ \text{sketch}[d,1] & \cdots & \text{sketch}[d,w] \end{bmatrix}$$

Among the user chosen parameters, ϵ controls the amount of tolerable error in the returned count and δ controls the probability with which the returned count is not within this acceptable error. These values of ϵ and δ determine the width and depth of the two-dimensional array respectively. To achieve the guarantees mentioned in the previous section, we set w=$\frac{2}{\epsilon}$ and d=$\log(\frac{1}{\delta})$. The depth d denotes the number of pairwise-independent hash functions employed by the algorithm and there exists an one-to-one correspondence between the rows and the set of hash functions. Each of these hash functions h_k:$\{x_1 \ldots x_N\} \rightarrow \{1 \ldots w\}$, $1 \leq k \leq d$ takes an item from the input stream and maps it into a counter indexed by the corresponding hash function. For example, $h_2(x) = 10$ indicates that the item "x" is mapped to the 10^{th} position in the second row of the sketch array. These d hash functions are chosen uniformly at random from a pairwise-independent family.

Initialize the entire sketch array with zeros.

Update Procedure: When a new item "x" with count c arrives[5], one counter in each row, as decided by its corresponding hash function, is updated by c. Formally, $\forall 1 \leq k \leq d$

$$\text{sketch[k,}h_k(x)\text{]} \leftarrow \text{sketch[k,}h_k(x)\text{]} + c$$

Query Procedure: Since multiple items can be hashed to the same counter, the frequency stored by each counter is an overestimate of the true count. Thus, to answer the point query, we consider all the positions indexed by the hash functions for the given item and return the minimum of all these values. The answer to Query(x) is: $\hat{c} = \min_k \text{ sketch}[k, h_k(x)]$.

Both update and query procedures involve evaluating d hash functions. Hence, both these procedures are linear in the number of hash functions. In our experiments (see Section5), we use d=3 similar to our earlier work (Goyal et al., 2010). Hence, the update and query operations take only constant time.

3.2 Properties

Apart from the advantages of being space efficient and having constant update and querying time, the CM sketch has other advantages that makes it attractive for scaling distributional similarity to the web:

1. Linearity: given two sketches s_1 and s_2 computed (using the same parameters w and d) over different input streams, the sketch of the combined data stream can be easily obtained by adding the individual sketches.
2. The linearity allows the individual sketches to be computed independent of each other. This means that it is easy to implement it in distributed setting, where each machine computes the sketch over a subset of the corpus.

3.3 Conservative Update

Estan and Varghese introduce the idea of conservative update (Estan and Varghese, 2002) in the context of networking. This can easily be used with CM Sketch (CU Sketch) to further improve the estimate of a point query. To update an item, w with frequency c, we first compute the frequency $\hat{c}$ of

[5] In our setting, c is always 1.

this item from the existing data structure and the counts are updated according to: $\forall 1 \leq k \leq d$

$$\text{sketch[k,}h_k(x)\text{]} \leftarrow \max\{\text{sketch[k,}h_k(x)\text{]}, \hat{c} + \text{c}\}$$

The intuition is that, since the point query returns the minimum of all the d values, we will update a counter only if it is necessary as indicated by the above equation. This heuristic avoids the unnecessary updating of counter values and thus reduces the error.

4 Efficient Distributional Similarity

To compute distributional similarity efficiently, we store counts in CU sketch. Our algorithm has three main steps:

1. Store approximate counts of all words, contexts, and word-context pairs in CU-sketch using fixed amount of counters.

2. Convert these counts into approximate PMI scores between word-context pairs. Use these PMI scores to store top K contexts for a word on the disk. Store these top K context vectors for every word stored in the sketch.

3. Use cosine similarity to compute the similarity between word pairs using these approximate top K context vectors constructed using CU sketch.

5 Word pair Ranking Evaluations

As discussed earlier, the DPs of words are used to compute similarity between a pair of words. We used the following four test sets and their corresponding human judgements to evaluate the word pair rankings.

1. **WS-353**: WordSimilarity-353[6] (Finkelstein et al., 2002) is a set of 353 word pairs.

2. **WS-203**: A subset of WS-353 containing 203 word pairs marked according to similarity[7] (Agirre et al., 2009).

3. **RG-65**: (Rubenstein and Goodenough, 1965) is set of 65 word pairs.

4. **MC-30**: A smaller subset of the RG-65 dataset containing 30 word pairs (Miller and Charles, 1991).

[6]http://www.cs.technion.ac.il/ gabr/resources/data/wordsim353/wordsim353.html

[7]http://alfonseca.org/pubs/ws353simrel.tar.gz

Each of these data sets come with human ranking of the word pairs. We rank the word pairs based on the similarity computed using DPs and evaluate this ranking against the human ranking. We report the spearman's rank correlation coefficient (ρ) between these two rankings.

5.1 Corpus Statistics

The Gigaword corpus (Graff, 2003) and a copy of the web crawled by (Ravichandran et al., 2005) are used to compute counts of all items (Table. 1). For both the corpora, we split the text into sentences, tokenize, convert into lower-case, remove punctuations, and collapse each digit to a symbol "0" (e.g. "1996" gets collapsed to "0000"). We store the counts of all words (excluding numbers, and stop words), their contexts, and counts of word-context pairs in the CU sketch. We define the context for a given word "x" as the surrounding words appearing in a window of 2 words to the left and 2 words to the right. The context words are concatenated along with their positions -2, -1, $+1$, and $+2$. We evaluate ranking of word pairs on three different sized corpora: Gigaword (GW), GigaWord + 50% of web data (GW-WB1), and GigaWord + 100% of web data (GW-WB2).

Corpus	**Sub set**	**GW**	**GW-WB1**	**GW-WB2**
Size (GB)	.32	9.8	49	90
# of sentences (Million)	2.00	56.78	462.60	866.02
Stream Size (Billion)	.25	7.65	37.93	69.41

Table 1: Corpus Description

5.2 Results

We compare our system with two baselines: Exact and Exact1000 which use exact counts. Since computing the exact counts of all word-context pairs on these corpora is not possible using main memory of only 8 GB , we generate context vectors for only those words which appear in the test set. The former baseline uses all possible contexts which appear with a test word, while the latter baseline uses only the top 1000 contexts (based on PMI value) for each word. In each case, we use a cutoff (of 10, 60 and 120) on the frequency of word-context pairs. These cut-offs were selected based on the intuition that, with more data, you get more noise, and not considering word-context pairs with frequency less than 120 might be a bet-

Data	GW			GW-WB1			GW-WB2		
Model	**Frequency cutoff**			**Frequency cutoff**			**Frequency cutoff**		
	10	60	120	10	60	120	10	60	120
	ρ			ρ			ρ		
WS-353									
Exact	.25	.25	**.22**	.29	.28	.28	.30	.28	.28
Exact1000	.36	**.28**	**.22**	**.46**	.43	.37	**.47**	**.44**	.41
Our Model	**.39**	**.28**	**.22**	-0.09	**.48**	**.40**	-0.03	.04	**.47**
WS-203									
Exact	.35	.36	.33	.38	.38	.37	.40	.38	.38
Exact1000	**.49**	**.40**	**.35**	**.57**	.55	**.47**	**.56**	**.56**	.52
Our Model	**.49**	**.39**	**.35**	-0.08	**.58**	**.47**	-0.06	.03	**.55**
RG-65									
Exact	**.21**	**.12**	**.08**	.42	**.28**	.22	.39	**.31**	.23
Exact1000	.14	.09	**.08**	**.45**	.16	.13	**.47**	.26	.12
Our Model	.13	**.10**	**.09**	-0.06	**.32**	.18	-0.05	.08	**.31**
MC-30									
Exact	.26	**.23**	**.21**	.45	.33	.31	.46	.39	.29
Exact1000	.27	.18	**.21**	**.63**	.42	.32	**.59**	**.47**	.36
Our Model	**.36**	.20	**.21**	-0.08	**.52**	**.39**	-0.27	-0.29	**.52**

Table 2: Evaluating word pairs ranking with Exact and CU counts. Scores are evaluated using ρ metric.

ter choice than a cutoff of 10. The results are shown in Table 2

From the above baseline results, first we learn that using more data helps in better capturing the semantic similarity between words. Second, it shows that using top (K) 1000 contexts for each target word captures better semantic similarity than using all possible contexts for that word. Third, using a cutoff of 10 is optimal for all different sized corpora on all test-sets.

We use approximate counts from CU sketch with $depth$=3 and 2 billion ($2B$) counters ('Our Model')[8]. Based on previous observation, we restrict the number of contexts for a target word to 1000. Table 2 shows that using CU counts makes the algorithm sensitive to frequency cutoff. However, with appropriate frequency cutoff for each corpus, approximate counts are nearly as effective as exact counts. For GW, GW-WB1, and GW-WB2, the frequency cutoffs of 10, 60, and 120 respectively performed the best. The reason for dependence on frequency cutoffs is due to the overestimation of low-frequent items. This is more pronounced with bigger corpus (GW-WB2) as the size of CU sketch is fixed to $2B$ counters and stream size is much bigger (69.41 billion) compared to GW where the stream size is 7.65 billion.

The advantages of using our model is that the sketch contains counts for all words, contexts, and word-context pairs stored in fixed memory of 8 GB by using only $2B$ counters. Note that it is not feasible to keep track of exact counts of all word-context pairs since their number increases rapidly with increase in data (see Fig. 1). We can use our model to create context vectors of size K for all possible words stored in the Sketch and computes semantic similarity between two words in O(K) time. In addition, the linearity of sketch allows us to include new incoming data into the sketch without building the sketch from scratch. Also, it allows for parallelization using the MapReduce framework. We can generalize our framework to any kind of association and similarity measure.

[8]Our goal is not to build the best distributional similarity method. It is to show that our framework scales easily to large corpus and it is as effective as exact method.

6 Conclusion

We proposed a framework which uses CU Sketch to scale distributional similarity to the web. It can compute similarity between any word pairs that are stored in the sketch and returns similarity between them in O(K) time. In our experiments, we show that our framework is as effective as using the exact counts, however it is sensitive to the frequency cutoffs. In future, we will explore ways to make this framework robust to the frequency cutoffs. In addition, we are interested in exploring this framework for entity set expansion problem.

Acknowledgments

We thank the anonymous reviewers for helpful comments. This work is partially funded by NSF grant IIS-0712764 and Google Research Grant Grant for Large-Data NLP.

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *NAACL '09: Proceedings of HLT-NAACL*.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Graham Cormode and Marios Hadjieleftheriou. 2008. Finding frequent items in data streams. In *VLDB*.

Graham Cormode and S. Muthukrishnan. 2004. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*.

Cristian Estan and George Varghese. 2002. New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4).

L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. 2002. Placing search in context: The concept revisited. In *ACM Transactions on Information Systems*.

J. Firth. 1968. A synopsis of linguistic theory 1930-1955. In F. Palmer, editor, *Selected Papers of J. R. Firth*. Longman.

Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. 2009. Streaming for large scale NLP: Language modeling. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian. 2010. Sketching techniques for Large Scale NLP. In *6th Web as Corpus Workshop in conjunction with NAACL-HLT*.

D. Graff. 2003. English Gigaword. Linguistic Data Consortium, Philadelphia, PA, January.

Z. Harris. 1985. Distributional structure. In J. J. Katz, editor, *The Philosophy of Linguistics*, pages 26–47. Oxford University Press, New York.

Abby Levenberg and Miles Osborne. 2009. Stream-based randomised language models for SMT. In *Proceedings of EMNLP*, August.

G.A. Miller and W.G. Charles. 1991. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.

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

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*.

H. Rubenstein and J.B. Goodenough. 1965. Contextual correlates of synonymy. *Computational Linguistics*, 8:627–633.

Florin Rusu and Alin Dobra. 2007. Statistical analysis of sketch estimators. In *SIGMOD '07*. ACM.

P.D. Turney and M.L. Littman. 2002. Unsupervised learning of semantic orientation from a hundred-billion-word corpus.

Peter D. Turney. 2008. A uniform approach to analogies, synonyms, antonyms, and associations. In *Proceedings of COLING 2008*.

Benjamin Van Durme and Ashwin Lall. 2009a. Probabilistic counting with randomized storage. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*.

Benjamin Van Durme and Ashwin Lall. 2009b. Streaming pointwise mutual information. In *Advances in Neural Information Processing Systems 22*.