# Streaming word similarity mining on the cheap

**Olof Görnerup**
RISE AI
SE-164 29 Kista, Sweden
olof.gornerup@ri.se

**Daniel Gillblad**
RISE AI
SE-164 29 Kista, Sweden
daniel.gillblad@ri.se

## Abstract

Accurately and efficiently estimating word similarities from text is fundamental in natural language processing. In this paper, we propose a fast and lightweight method for estimating similarities from streams by explicitly counting second-order co-occurrences. The method rests on the observation that words that are highly correlated with respect to such counts are also highly similar with respect to first-order co-occurrences. Using buffers of co-occurred words per word to count second-order co-occurrences, we can then estimate similarities in a single pass over data without having to do prohibitively expensive similarity calculations. We demonstrate that this approach is scalable, converges rapidly, behaves robustly under parameter changes, and that it captures word similarities on par with those given by state-of-the-art word embeddings.

## 1 Introduction

Word similarities play an integral part in many natural language processing applications. Improving similarity estimates will therefore in turn potentially improve a broad range of areas, including word alignment (Songyot and Chiang, 2014), query expansion (Diaz et al., 2016), simplification (Biran et al., 2011), document classification (Arras et al., 2017), lexical substitution (McCarthy and Navigli, 2009) and many more.

The prevalent approach to estimate similarities is to first embed words in a vector space using techniques such as word2vec (Mikolov et al., 2013a,b) and GloVe (Pennington et al., 2014), and then calculate the similarities between words as the similarities between corresponding vectors. Finding all significant similarities among a set of words in this way, however, is computationally demanding due to the large number of pairwise similarity calculations involved (scaling as the square of the vocabulary size at worst). All-to-all similarity calculations are in particular strenuous, if at all feasible, in a streaming setting due to tight latency and memory constraints.

In this paper we address this by proposing a method that *finds significant similarities without calculating any similarities*. This seemingly contradictory feat is possible by explicitly counting *second-order* co-occurrences (SOCOs for short) and calculating correlations with respect to these. Two words $w$ and $v$ are said to have a SOCO if there is a third word $u$ with whom both $w$ and $v$ co-occur (not necessarily together). For example, if the words *hot* and *coffee* co-occur at some point in a corpus or stream, and *hot* and *tea* co-occur at some other point in the same corpus or stream, then *coffee* and *tea* have a SOCO relation since they both co-occur with *hot*. The key observation then, as depicted in Fig. 1, is that *words that are highly correlated with respect to second-order co-occurrences are highly similar with respect to first-order co-occurrences*. This relation enables us to avoid pairwise similarity calculations altogether and instead acquire similarities directly from the SOCO counts.

The contribution of this paper is mainly twofold. Firstly, we introduce an operational definition of SOCO probabilities. To the best of our knowledge, this has not been done before in this explicit manner. Secondly, we apply this definition to efficiently estimate word similarities from streams. In practice we achieve this by keeping small buffers of co-occurred words per context word, and then incrementing SOCO counts of new co-occurred words and those in the buffer. Importantly, this enables us to pass over data only once. To ensure scalability in memory usage and runtime, approximate SOCO counts are then maintained in a count-min sketch table (Cormode and Muthukrishnan, 2005) that keeps the algorithm lightweight.

The benefits of our approach are not only com-



*Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1425–1434
Brussels, Belgium, October 31 - November 4, 2018. 

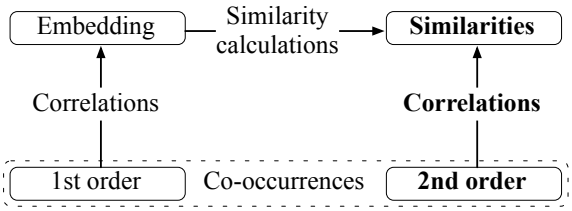


Figure 1: Diagram that describes the relationship between 1st and 2nd order co-occurrences. Instead of first embedding words with regard to 1st order co-occurrences and then calculating pairwise similarities between word vectors, we acquire similarities by directly correlating words with respect to their explicit 2nd order co-occurrences (encompassing labels in boldface).

putational in allowing us to avoid both multi-pass batch processing and the quadratic time complexity that comes with pairwise similarity calculations. Since the method is based on simple counting of SOCOs, it is also completely transparent and interpretable. By comparison, most first-order word embeddings – word2vec and GloVe are again good examples – are relatively opaque and difficult to interpret. The simplicity of our method also makes it straightforward to implement while having only a handful of parameters to tune compared to prevalent embedding methods that typically involve a large number hyperparameters.

The remainder of this paper is outlined as follows: Next we will relate our proposed method to current approaches for calculating word similarities. The method is described in Sec. 3, followed by a complexity analysis in Sec. 4 where we theoretically and experimentally show that our approach indeed is suitable for stream mining. In Sec. 5, we evaluate the method with respect to convergence, accuracy and parameter sensitivity. The paper is concluded in Sec. 6 with a summary and a discussion on possible future directions.

## 2 Related work

The explicit use of second-order co-occurrences has so far gained little attention in word similarity mining. The typical approach is rather to express word similarities as similarities between vector representations that in turn are based on first-order co-occurrences (this is also the case in (Islam and Inkpen, 2006), despite the name of their approach[1]).

[1]In general, the term *second-order co-occurrence* is sometimes used in NLP to describe second-order representations based on first-order word vectors, such as in (Schütze, 1998). This does not involve explicit SOCO counts as in our case though.

There is a large body of work in this area, predominantly based on Harris' distributional hypothesis (Harris, 1954), from seminal approaches such as LSA/LSI (Deerwester et al., 1990), HAL (Lund and Burgess, 1996) and Random indexing (Kanerva et al., 2000) and onward. See (Levy et al., 2015) for an extensive review. A common approach then is to represent words in terms of co-occurrence correlations – e.g. using Pointwise mutual information (PMI) (Church and Hanks, 1990) and variants thereof (Levy et al., 2015) – either explicitly or through dimensionality reduction (Pennington et al., 2014). Another prevalent approach is to generate vector representations based on prediction tasks, where words are predicted from their contexts or vice versa; Continuous bag of words and Skip-gram (Mikolov et al., 2013a,b) are prominent examples. However, these methods are batch-based and typically require multiple passes over data (3 to 50 training epochs in the case of word2vec, for instance (Mikolov et al., 2013b)). Approaches that are more suitable for streaming data have also been developed, e.g. for calculating the most PMI-correlated words per word (Durme and Lall, 2009) and, more recently, neural network methods (primarily based on Skip-gram) adapted to enable incremental updating (Luo et al., 2015; Kaji and Kobayashi, 2017; Bamler and Mandt, 2017; Peng et al., 2017). Note though that in all of the above cases vector representations capture *first-order* co-occurrences. When using PMI for example, the correlation between two words is high if they co-occur more frequently than expected from chance, but high correlation does not equate high similarity (consider the words *red* and *wine* for example). Estimating word similarities would therefore require the extra step of calculating similarities between vectors, something that our approach bypasses by explicitly counting SOCOs.

## 3 Method

### 3.1 Second-order co-occurrences

We define the SOCO probability of words $w$ and $v$ as

$$P(w:v) := \sum_{u \in \mathcal{V}} P(u)P(w|u)P(v|u), \quad (1)$$

where $w:v$ denotes a SOCO and $\mathcal{V}$ is the vocabulary. That is, $P(w:v)$ is the probability that two

randomly selected words co-occurring with a randomly selected word $u$ are $w$ and $v$. Since

$$P(w:v) = \sum_{u \in \mathcal{V}} P(w,u)P(v|u) \quad (2)$$
$$= \sum_{u \in \mathcal{V}} P(w)P(u|w)P(v|u),$$

an alternative interpretation of SOCO is the chain of randomly selecting a word $w$, one of its co-occurring words $u$, and in turn one of *its* co-occurring words $v$.

### 3.2 Correlation measure

Our ansatz is that two words have a high degree of similarity if they are highly correlated with respect to SOCO, since this would imply that the words are relatively interchangeable in their respective contexts. We quantify a SOCO correlation using standard pointwise mutual information (PMI):

$$M(w:v) = \log_2 \frac{P(w:v)}{P(w)P(v)}, \quad (3)$$

where the denominator is the SOCO probability of $w$ and $v$ given that they are independent of $u$, since then

$$P(w:v) = \sum_{u \in \mathcal{V}} P(u)P(w)P(v). \quad (4)$$

### 3.3 Estimating correlations

Making the simplifying assumptions that a stream is stationary and that co-occurrence correlations decay rapidly with word-to-word distance in the stream (or corpus), we can estimate Eq. 3 in one pass using counters of word occurrences and SOCOs. See Fig. 2 for a schematic overview and Algorithm 1 for a detailed description of our approach.

#### 3.3.1 Co-occurrence buffers

Approximate SOCO counts are maintained by keeping small buffers (on the order of 1 to 10 words) of previously observed words with a given context. The context of a word is here given by the position relative to the word (say, -1 for the preceding word) and the word occupying that position. For example, if we observe the word *fox* in the sequence

*The quick brown fox jumps over the lazy dog.*

*fox* is added to the buffer of previously observed words with context (-1, *brown*) (*bear*, *bag* and *eyes*

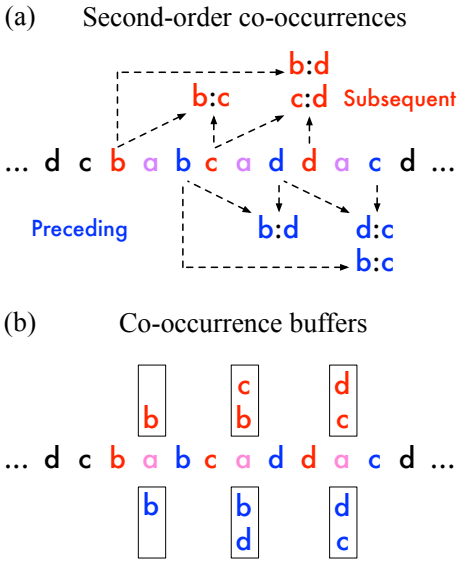


Figure 2: (a) Words that share a context word (the word $a$ in this example) at a given position (as a preceding and subsequent word, in blue and red, respectively) are said to co-occur to the second order. For example, $b$ and $c$ have a SOCO relation $b:c$ since they both have $a$ as a subsequent word. (b) We count second-order co-occurrences by keeping buffers of co-occurring words for a given context. The buffer for context (1, $a$) is shown above the sequence (i.e., when $a$ occurs as a subsequent word) and (-1, $a$) below the sequence ($a$ as a preceding word). Both buffers (implemented as queues, where the most recent observations are stored) are limited to two words in this example. Before a word is added to a buffer, the second-order co-occurrence counts of the word and the words in the buffer are incremented. For instance, the counts of $b:d$ and $c:d$ are incremented before the buffer $[c, b]$ is updated to $[d, c]$.

perhaps). Note that *fox* then has a SOCO relation with the words in the buffer. The SOCO counters of (*fox*, $w$) for all words $w$ in the buffer are therefore incremented (for (*fox*, *bear*) for instance) before *fox* is added to the buffer. If the buffer is full – we cap the number of prior words stored – the oldest word is discarded prior to adding the new one.

The same procedure is performed for all context positions in a sliding window of a given length (e.g., for positions $\{-2, -1, 1, 2\}$ if we consider symmetric contexts in a five-word window). By simultaneously estimating word frequencies by counting current words we can in this way incrementally maintain estimates of $M(w:v)$.

#### 3.3.2 Probabilistic counting

Since the number of SOCOs may be very large, we keep approximate counts of these using a count-min sketch table (Cormode and Muthukrishnan,

2005). The table, denoted $d$, has $h$ rows and $g$ columns, where the rows are associated with $h$ pairwise independent hash functions, $f_i$, and where entries are initialized to 0. When a SOCO $w:v$ for words $w$ and $v$ is observed, each hash function maps $w:v$ to an index $f_i(w:v)$ of row $i$. Entry $(i, f_i(w:v))$ is then incremented by 1.

After populating $d$ the approximate count of $w:v$ is given by the minimum count of the entries $(i, f_i(w:v))$:

$$\tilde{c}(w:v) = \min_i d(i, f_i(w:v)). \quad (5)$$

Note that $\tilde{c}$ may overestimate the true count $c(w:v)$ due to hash collisions (this is indeed what keeps the data structure sublinear with respect to the number of SOCOs). To reduce overestimations we can employ conservative updates by only updating an entry $(i, f_i(w:v))$ if it is exceeded by the current incremented approximate SOCO count (Goyal et al., 2012):

$$d(i, f_i(w:v)) \leftarrow \max\{d(i, f_i(w:v)), \tilde{c}(w:v)\}. \quad (6)$$

This is the approach used in all experiments presented below.

#### 3.3.3 Top-$k$ correlations

Rather than storing all SOCO correlations, which is infeasible for large vocabularies and wide context ranges, we keep track of the $k$ most correlated words for each word in the vocabulary. We achieve this scalably by adapting the approach proposed by Durme and Lall for calculating first-order co-occurrence correlations in streams (Durme and Lall, 2009). We keep track of occurred SOCOs in non-overlapping meta-windows (on the order of $10^5$ to $10^7$ tokens long). For each meta-window, we calculate PMIs for occurred SOCOs using exact word counts (having a comparably small memory footprint) and approximate SOCO counts, and then update priority queues with the most similar words per word accordingly. In this way the number of SOCOs stored is kept approximately constant as we consume the stream.

**Algorithm 1** Estimate $M(w:v)$ for stream $[s_1, s_2, ..., s_m]$, vocabulary $\mathcal{V}$, context range $n$, buffer capacity $r$, meta-window size $N$ and count-min (C-m) table with $h$ rows and $g$ columns. Note that $m$ may approach infinity.

$l \leftarrow \{-n, ..., -1, 1, ..., n\}$ {context positions}
$c(w) \leftarrow 0; w \in \mathcal{V}$ {word counts}
$d_{i,j} \leftarrow 0; i = 1, ..., g, j = 1, ..., h$ {C-m table}
$t_w \leftarrow 0$ {total word count}
$t_s \leftarrow 0$ {total SOCO count}
$q(i, w) \leftarrow \emptyset; w \in \mathcal{V}, i \in l$ {context queues}
$\mathcal{S}(w) \leftarrow \emptyset; w \in \mathcal{V}$ {similar word queues}
$\mathcal{P} \leftarrow \emptyset$ {observed SOCOs}
**for** $i = n + 1$ to $m - n$ **do** {consume stream}
  $c(s_i) \leftarrow c(s_i) + 1$
  $t_w \leftarrow t_w + 1$
  **for** $j \in l$ **do**
    **for** $v \in q(j, s_{i+j}), v \neq s_i$ **do**
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{s_i:v\}$
      $d(s_i:v) \leftarrow d(s_i:v) + 1$
      $t_s \leftarrow t_s + 1$
    **end for**
    **if** $|q(j, s_{i+j})| = r$ **then** {buffer full}
      $q(j, s_{i+j})$.dequeue() {discard oldest}
    **end if**
    $q(j, s_{i+j})$.enqueue($s_i$) {add current}
  **end for**
  **if** $i \bmod N = 0$ **then** {meta-window ends}
    **for** $w \in \mathcal{V}$ **do**
      $P(w) \leftarrow c(w)/t_w$
    **end for**
    **for** $w:v \in \mathcal{P}$ **do**
      $P(w:v) \leftarrow d(w:v)/t_s$
      $M = \log_2[P(w:v)/(P(w)P(v))]$
      Update $\mathcal{S}(w)$ with $v$ and priority $M$
    **end for**
    $\mathcal{P} \leftarrow \emptyset$ {clear observed SOCOs}
  **end if**
**end for**

## 4 Complexity analysis

### 4.1 Time

For each token in the stream, updating word counts takes constant time using an associative array. The algorithm also goes through $2n$ context words, where $n$ is the context range. For each context word, the count-min table that stores approximate SOCO counts is updated at most $r$ times, where $r$ is the maximum buffer size. Since each update takes $O(h)$ time for a count-min table with $h$ rows (a hash function is called for each row), the time complexity for updating counters is $O(nrh)$. Updating a co-occurrence buffer, i.e. by inserting and (possibly) deleting a word index in a queue, takes $O(1)$ time, and so the total time complexity per token when consuming a meta-window is $O(nrh)$.

Between meta-windows, we update the top cor-

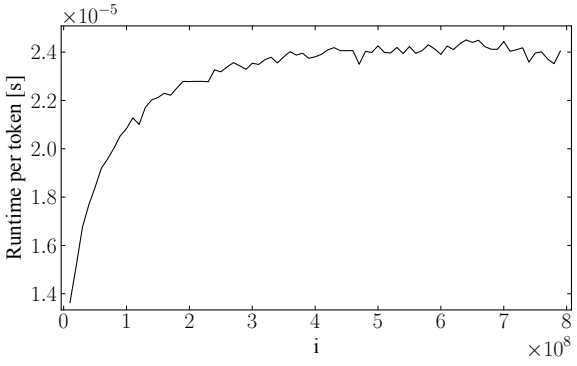


Figure 3: Runtime per token at stream position $i$ (Python implementation run on a MacBook Pro with a 2.8 GHz Intel Core i7 processor and 16 GB of memory).

related words per word. We first update frequency estimates of words, which takes $O(|\mathcal{V}|)$ time. For each SOCO that has been observed in the meta-window, $|\mathcal{P}|$ of them, we update the priority queues with the most correlated words of the words involved in the SOCOs. Each such update takes $O(\log k)$, where the parameter $k$ is the number of most correlated words of a word.

Altogether, the time complexity is hence $O(nrh + \frac{|\mathcal{V}|+|\mathcal{P}|\log k}{N})$ per token. Note that $n, r, h, k \ll |\mathcal{V}|, |\mathcal{P}|$, where $n$, $r$, $h$ and $k$ are all small fixed parameters ($n \sim 1$ to 10, $r \sim 10$, $k \sim 10$ to 100 and $h \sim 10$ typically). Given that the stream is roughly stationary (which has been the case in our experiments), the size of $\mathcal{P}$ is approximately constant. With a fixed vocabulary and $N$, the runtime per token is therefore also kept approximately constant. We confirm this experimentally, see Fig. 3, by running the algorithm on English Wikipedia (as of March 7, 2015) using a context range of one (window size three), a vocabulary constituted by the $10^4$ most frequent words, meta-windows of length $10^7$ and five-word co-occurrence buffers, a count-min table with 8 rows and $3.4 \cdot 10^7$ columns, and where the 10 most similar words per word are stored. As expected, the runtime per token increases initially as the co-occurrence buffers are filled up, and then converges towards a constant value.

### 4.2 Space

With regard to memory the algorithm requires that we keep a co-occurrence buffer for each word in the vocabulary and for each context position. The space complexity of the buffers is therefore $O(nr|\mathcal{V}|)$. In addition, there are $|\mathcal{V}|$ word counters and priority queues of size $k$ for storing top-$k$ similar words, a count-min sketch table with $gh$ entries, where the fixed parameter $g$ is the number of columns in the count-min table, and a index pair set of size $|\mathcal{P}|$ for storing occurred SOCOs that have occurred in a meta-window. The total space required is hence $O((nr + k)|\mathcal{V}| + gh + |\mathcal{P}|)$.

For example, assume we have a vocabulary with a million words, a context range of 5 (i.e., window size 5+1+5), a buffer size of 10, a count-min table with $3 \cdot 10^8$ columns and 8 rows, and that we keep the 10 most similar words per word. We then need to store $10^6 \cdot 2 \cdot 5 \cdot 10 = 10^8$ items in the co-occurrence buffers. If each item requires 4 bytes (a word index constituted by an unsigned integer), the buffers take up 400 MB of space. Add to that another 4 MB for the word counters ($10^6$ of them à 4 bytes), 40 MB ($10^6 \cdot 4 \cdot 10$ bytes) for the top 10 similar word indices per word and 9.6 GB ($8 \cdot 3 \cdot 10^8 \cdot 4$ bytes) for the count-min sketch table. Set the meta-window length so that $|\mathcal{P}| \leq 7.4 \cdot 10^8$ (taking up a bit less than 6 GB of memory at most) and we end up with a total footprint of approximately 16 GB – a memory requirement even met by many present-day laptops.

## 5 Evaluation

### 5.1 Examples

In Table 1 we show a set of examples of the most SOCO-correlated words per word as output by the algorithm (using Wikipedia, a context range of 2, buffer size of 10 and a count-min table with 8 rows and $2.7 \cdot 10^8$ columns). Although the examples are anecdotal, they illustrate that the method manages to mine word similarities that make intuitive sense, such that *Wednesday* is most similar to other days in the week (interestingly, it is most similar to adjacent days, *Tuesday* and *Thursday*) and *yellow* is most similar to other colors, etc.

### 5.2 Convergence

It is crucial that the method converges within a reasonable amount of time in order for it to be of practical use. To test this we run the algorithm and measure how the sets of top-$k$ similar words change as the stream progresses. We quantify these changes using the Jaccard index between a word set of a word $w$ at position $i$ in the stream, $\mathcal{S}_i(w)$, and the corresponding word set at $\Delta i$ tokens prior,

| musician | increasing | croatian | wednesday | scholar | hermann | coventry | yellow |
|---|---|---|---|---|---|---|---|
| singer | reducing | yugoslav | thursday | scholars | heinrich | leicester | purple |
| pianist | growing | serbian | tuesday | translator | friedrich | norwich | pink |
| songwriter | increased | slovenian | monday | playwright | wilhelm | stoke | orange |
| guitarist | reduces | croatia | friday | philosopher | georg | swansea | blue |
| rapper | reduce | slovak | saturday | poet | wolfgang | cardiff | red |

Table 1: Examples of the most correlated words per word with respect to second-order co-occurrence.

| | SIMLEX | SIMVERB | MT-287 | MT-771 | WS-353 |
|---|---|---|---|---|---|
| SOCO | 0.41 | 0.25 | 0.59 | 0.56 | 0.71 |
| FOCO | 0.35 | 0.23 | 0.65 | 0.59 | 0.66 |
| GLOVE | 0.31 | 0.18 | 0.61 | 0.57 | 0.63 |
| CBOW | 0.34 | 0.22 | 0.66 | 0.57 | 0.69 |
| SGM | 0.41 | 0.32 | 0.67 | 0.60 | 0.72 |
| Coverage | 0.99 | 0.96 | 0.95 | 0.98 | 0.98 |

Table 2: Top rows: Spearman's rank correlation coefficients for different methods and benchmarks. Bottom row: fractions of benchmark word pairs covered by the methods and the corpus.

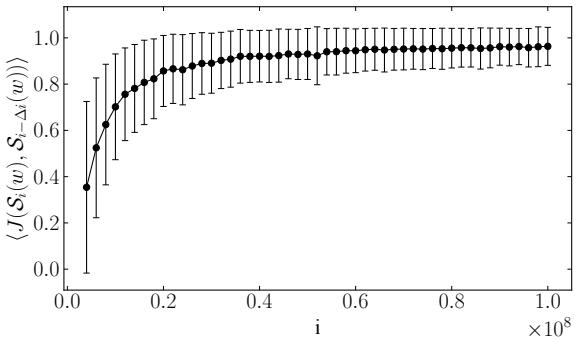


Figure 4: Average change of top-$k$ word sets over words at stream position $i$. Standard deviation shown by error bars.

$\mathcal{S}_{i-\Delta i}(w)$. That is,

$$J(\mathcal{S}_i(w), \mathcal{S}_{i-\Delta i}(w)) = \frac{|\mathcal{S}_i(w) \cap \mathcal{S}_{i-\Delta i}(w)|}{|\mathcal{S}_i(w) \cup \mathcal{S}_{i-\Delta i}(w)|}, \quad (7)$$

where $J = 1$ if there is no change and the sets are identical. As seen in Fig. 4, the top-$k$ sets converge as both the mean of $J(\mathcal{S}_i(w), \mathcal{S}_{i-\Delta i}(w))$ over words $w$ tends towards one while the standard deviation decreases. In this experiment, applied on Wikipedia and a vocabulary of $10^4$ words, the top ten words per word were stored. Further, $\Delta i = 10^6$ tokens, contexts of range one, a count-min table with 8 rows and $3.4 \cdot 10^7$ columns, and co-occurrence buffers of size five were used.

### 5.3 Accuracy

To quantitatively evaluate the accuracy of the method, we use a collection of established word similarity benchmarks: SimLex-999 (SIMLEX) (Hill et al., 2015), SimVerb (SIMVERB) (Gerz et al., 2016), MTurk-287 (MT-287) (Radinsky et al., 2011), MTurk-771 (MT-771) (Halawi et al., 2012) and WordSim (WS-353) (Finkelstein et al., 2001). Each of these benchmarks contains a set of word pairs and their similarities as judged by human annotators. Comparing these word rankings with rankings given by Eq. 3, we get an indication of how well our method captures human notions of similarity and relatedness. The agreement is quantified with the standard Spearman's rank correlation coefficient.

The results are also compared to the ranking agreements for popular word embeddings – GloVe (Pennington et al., 2014) (GLOVE), Continuous Bag of words (CBOW) and Skip-gram (SGM) (Mikolov et al., 2013a,b) – as well as for the pointwise mutual information between regular first-order co-occurrences (FOCO). In all these cases, word similarities are given by the cosine similarity,

$$\sigma(v_i, v_j) = \frac{v_i \cdot v_j}{|v_i|_2 |v_j|_2}, \quad (8)$$

where $v_i$ and $v_j$ are vectors associated with words $i$ and $j$. All word embeddings are in 300 dimen-

sions. Remaining parameters are set to default values, with the exception that no explicit word count thresholding is used in order to avoid loss of benchmark words. (Applying thresholds at 3-5 occurrences gives comparable results as those reported here, but for fewer word pairs.) For all benchmarks the One billion word corpus[2] (Chelba et al., 2013) is used with a vocabulary consisting of the $3 \cdot 10^4$ most frequent words, except the Stanford CoreNLP stopwords.[3] A context range of 2, count-min tables with 8 rows and $3.4 \cdot 10^7$ columns, and a buffer size of 10 are used throughout. Since many word pairs in the benchmarks are dissimilar we keep all SOCO correlations for the benchmark word pairs in this experiment. A small fraction of benchmark words are either not present in the corpus or captured by SOCO or the embedding methods due to sampling. To ensure a fair comparison between methods, only word pairs that are represented by all approaches are therefore considered.

The results are summarized in Table 2, where we note that the coverage (the fraction of benchmark pairs represented) is high, from 95% for MT-287 to 99% for SIMLEX. The relative performance of SOCO varies over benchmarks: in two out of five cases the performance is roughly on par with SGM, and compared to GLOVE, CBOW and FOCO, our method performs best in three out of five benchmarks. Thus the overall picture is that our approach indeed is able to capture meaningful similarity relations, and that it performs comparably to regular first-order word embedding methods.

### 5.4 Parameter sensitivity

The algorithm has two key parameters: the co-occurrence buffer capacity and the size of the count-min sketch table. Since the approximation errors induced by the latter is thoroughly analyzed in (Cormode and Muthukrishnan, 2005) we will here focus on how the buffer size influences similarity accuracy. Using the same corpus and benchmark suite as in Sec. 5.3, we evaluate how the accuracy varies with the buffer capacity. To again make a fair comparison, we then only include those benchmark word-pairs that are covered in all experiments.

As seen in Fig. 5, the method is insensitive to the buffer capacity size as the accuracy stays approximately constant for buffers of sizes larger than 3. This is also the case in relative terms: see Fig. 6

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

[3]https://stanfordnlp.github.io/CoreNLP/

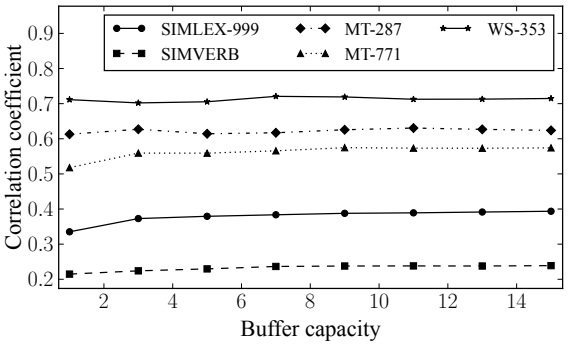


Figure 5: Spearman's rank correlation coefficients for different benchmarks and buffer capacities.

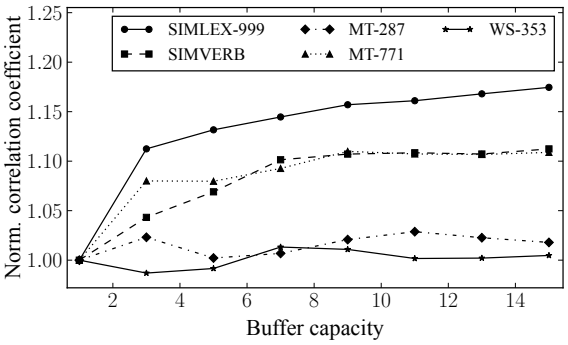


Figure 6: Relative Spearman's rank correlation coefficients with respect to correlations at buffer capacity 1 for different benchmarks and buffer capacities.

where we plot the accuracy relative to the accuracy for buffers of size 1. The improvement is then largest for SIMLEX-999, where going from buffers of size 1 to 15 yields an approximate difference of 17% in accuracy. The relative accuracy, however, varies little from buffer capacity 3 and upward. With regard to coverage, see Fig. 7, the buffer capacity has a significant effect up until buffer size 9, after which the coverage settles at around 95-99%.

We can conclude that the method is robust with regard to buffer capacity, resulting in predictable and smooth changes in output, and that it suffices to keep small buffers to maintain both accuracy and coverage. Since the buffers in effect tend to store the most frequent co-occurrences per word, these results indicate that it is possible to accurately estimate similarities using only salient co-occurrences. This is also supported by Polajnar and Clark's finding (2014) that only a handful of the most frequent context words yields the best results when estimat-

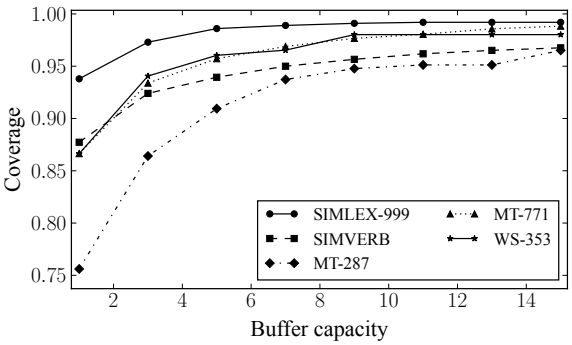


Figure 7: Coverage for different benchmarks and buffer capacities.

ing similarities from co-occurrence frequencies.

## 6 Conclusions

We have presented a method for estimating word similarities from corpora or streams using an explicit notion of SOCO. Our approach is simply to count such co-occurrences and calculate correlations between words with respect to these counts. Words that are highly correlated are then also highly similar with respect to first-order co-occurrences. By using co-occurrence buffers and count-min sketches for estimating SOCO counts, the method keeps both the runtime per token and memory usage constant while only needing one pass over data. These properties makes our approach ideal for low-cost stream mining.

Despite its simplicity and modest computational requirements, benchmark experiments show that the method performs comparably to calculating similarities between best-in-class word vectors. This not only makes our approach a feasible alternative for calculating word similarities on the cheap, but in some cases it may be the only viable option. Consider for instance an embedded system in a decentralized machine learning or edge computing scenario. Then real-time computing constraints and scarce memory would rule out both multi-pass word embeddings and pairwise similarity calculations in favor of similarities readily available from SOCO counts.

There are numerous possible future directions, exploring correlation measures other than PMI being one. Also, by grouping words in concurrence with finding top-$k$ similar words per word, an extended method could be used for word clustering. Possible ways to find groups of inter-similar words – constituting abstract concepts (Görnerup et al., 2017) – is then to use label propagation on a graph (Raghavan et al., 2007) (with words constituting vertices and the top-$k$ similar words directed edges), or agglomerative hierarchical clustering. How to do this efficiently and scalably is currently under study. Moreover, in this paper we have exclusively considered text data and word similarities. The method, however, is domain-agnostic and may be applied on other types of data, in and beyond the NLP domain. There is a wide range of potential application areas, presumably in everything from biology and physics to social science and economics – in essence in any domain where objects co-occur and where these co-occurrences carry some relevant information or meaning.

## Acknowledgments

The authors thank the anonymous reviewer for bringing to their attention the link between most frequent co-occurrences and the sufficiency of small buffer sizes.

This work was funded by the Swedish Knowledge Foundation through the BIDAF project and by the Swedish Foundation for Strategic Research through the Continuous Deep Analytics project.

## References

Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. 2017. "What is Relevant in a Text Document?": An Interpretable Machine Learning Approach. *PLOS ONE*.

Robert Bamler and Stephan Mandt. 2017. Dynamic word embeddings. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 380–389.

Or Biran, Samuel Brody, and Noémie Elhadad. 2011. Putting It Simply: A Context-aware Approach to Lexical Simplification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 496–501. Association for Computational Linguistics.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

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

Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms*, 55(1):58–75.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Fernando Diaz, Bhaskar Mitra, and Nick Craswell. 2016. Query expansion with locally-trained word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 367–377. Association for Computational Linguistics.

Benjamin V. Durme and Ashwin Lall. 2009. Streaming Pointwise Mutual Information. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 1892–1900. Curran Associates, Inc.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 406–414. ACM.

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity. In *EMNLP*.

Olof Görnerup, Daniel Gillblad, and Theodore Vasiloudis. 2017. Domain-agnostic discovery of similarities and concepts at scale. *Knowledge and Information Systems*, 51(2):531–560.

Amit Goyal, Hal Daumé, III, and Graham Cormode. 2012. Sketch Algorithms for Estimating Point Queries in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1093–1103. Association for Computational Linguistics.

Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. 2012. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1406–1414. ACM.

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

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. Simlex-999: Evaluating semantic models with genuine similarity estimation. *Comput. Linguist.*, 41(4):665–695.

A. Islam and D. Inkpen. 2006. Second order co-occurrence PMI for determining the semantic similarity of words. In *LREC 2006*, pages 1033–1038.

Nobuhiro Kaji and Hayato Kobayashi. 2017. Incremental skip-gram model with negative sampling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 363–371. Association for Computational Linguistics.

P. Kanerva, J. Kristofersson, and A. Holst. 2000. Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, volume 1036.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225.

Kevin Lund and Curt Burgess. 1996. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208.

Hongyin Luo, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2015. Online learning of interpretable word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1687–1692.

Diana McCarthy and Roberto Navigli. 2009. The english lexical substitution task. *Language Resources and Evaluation*, 43(2):139–159.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

Hao Peng, Jianxin Li, Yangqiu Song, and Yaopeng Liu. 2017. Incrementally learning the hierarchical softmax function for neural language models. In *AAAI*, pages 3267–3273. AAAI Press.

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

Tamara Polajnar and Stephen Clark. 2014. Improving distributional semantic vectors through context selection and normalisation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 230–238.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A word at a time: Computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 337–346. ACM.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106.

Hinrich Schütze. 1998. Automatic word sense discrimination. *Journal of Computational Linguistics*, 24:97–123.

Theerawat Songyot and David Chiang. 2014. Improving word alignment using word similarity. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1840–1845, Doha, Qatar. Association for Computational Linguistics.