S. Khalife, D. Gon¸calves, Leo Liberti, Distance geometry for word representations and applications, Journal of Computational Mathematics and Data Science, 6:100073, 2023

Distance Geometry for Word Representations and Applications[☆]

Sammy Khalife[a,*], Douglas S. Gonçalves[b], Leo Liberti[c]

[a]*Department of Applied Mathematics and Statistics, Johns Hopkins University*
[b]*MTM/CFM, Universidade Federal de Santa Catarina, 88040-900 Florianópolis, Brazil*
[c]*LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France*

Abstract

Many machine learning methods used for the treatment of sequential data often rely on the construction of vector representations of unitary entities (e.g. words in natural language processing, or k-mers in bioinformatics). Traditionally, these representations are constructed with optimization formulations arising from co-occurrence based models. In this work, we propose a new method to embed these entities based on the Distance Geometry Problem: find object positions based on a subset of their pairwise distances or inner products. Considering the empirical Pointwise Mutual Information as a surrogate for the inner product, we discuss two Distance Geometry based algorithms to obtain word vector representations. The main advantage of such algorithms is their significantly lower computational complexity in comparison with state-of-the-art word embedding methods, which allows us to obtain word vector representations much faster. Furthermore, numerical experiments indicate that our word vectors behave quite well on text classification tasks in natural language processing as well as regression tasks in bioinformatics.

Keywords: Distance geometry, Optimization, Computational Complexity, Co-occurrence data, Unsupervised Learning

[☆]Partial support by ANR project ANR-19-CE45-0019 (multiBioStruct) is gratefully acknowledged. DSG is thankful to CAPES/Print Process 88887.465828/2019-00.

[*]Corresponding author

Email addresses: khalife.sammy@jhu.edu (Sammy Khalife), douglas@mtm.ufsc.br (Douglas S. Gonçalves), liberti@lix.polytechnique.fr (Leo Liberti)

1. Introduction

Given a finite set $\mathcal{V}$ of symbols called *vocabulary*, the choice of vector or matrix representations of sequences of values in the vocabulary poses a central problem in data science. Such a representation is necessary to represent strings over $\mathcal{V}$ as input to Machine Learning (ML) techniques. This need is particularly strong in ML applications such as Natural Language Processing (NLP), which uses natural language sentences as input, or bioinformatics, when it is concerned with protein sequences — i.e. amino acid sequences — as input.

A sequence s of symbols in $\mathcal{V}$ naturally possesses a trivial vector representation, namely the *incidence vector* of the sequence in $\mathcal{V}$ (which actually represents the underlying set of s), and the so-called "one-hot encoding", a $|\mathcal{V}| \times |s|$ matrix (where $|s|$ is the sequence length) bearing a 1 at position (i, j) if and only if the i-th word in $\mathcal{V}$ appears in the j-th position in the sentence s, and 0 otherwise. We note that the incidence vector is obtained from the one-hot encoding by applying the $\max$ operator on each row.

Because of their size and simplicity, these trivial representations are rarely sufficient for today's needs. The construction of more compact and informative representations is often achieved using unsupervised learning approaches. The obtained representations — also called *embeddings* in ML — are grouped in two categories: *static* and *contextual* embeddings. A static embedding is a function $f : x \in \mathcal{V} \mapsto v \in \mathbb{R}^K$, where, usually, $K \ll |\mathcal{V}|$ holds. Traditionally, f is determined using co-occurrence data from large training corpora [1, 2]. In this case, co-occurences of words are determined using every "window" (i.e. subsequence) of w consecutive elements of the sequence, called *tokens*. Contextual embeddings, which are popular in the Deep Learning community, are functions defined over the set of sequences $\mathcal{V}^*$ engendered by the vocabulary $\mathcal{V}$. Specifically, a contextual embedding is a mapping $f : \mathcal{V}^* \mapsto \mathbb{R}^{N \times K}$ where N represents the maximum number of words of a sequence (usually $N \leq 10^3$ [3]). Unlike static embeddings, the restriction of contextual embeddings to a single symbol may yield different vectors for the same symbol when it appears in different sequences.

In this article, we present, adapt and apply Distance Geometry (DG) based methods in order to develop faster word vectors construction algorithms. We provide a rigorous computational complexity analysis of the proposed methods and compare with those of standard methods for static word representation. Furthermore, we show empirically that word vectors obtained by our DG-based methods behave well on extrinsic tasks, such as text classification and regression, in natural language processing and bioinformatics.

To the best of our knowledge, apart from [4], the DG paradigm has not been exploited for determining word vector representations. Differently from [4], where the graph-of-words model [5] was considered and the corresponding graph realization problem addressed with nonlinear programming methods and heuristics, here we consider a probabilistic model relating pointwise mutual information (PMI) and inner products, and solve the resulting Distance Geometry Problem (DGP) with methods based on Euclidean Distance Matrices [6] and trilateration [7].

The rest of this paper is organized as follows. In Section 2 we introduce the problem of determining word representations in natural language processing. Then, Section 3 reviews some useful DG concepts and presents two algorithms for DGP. Section 4 describes the word co-occurrence model and explain how the DG-based algorithms from Section 3 can be used to build word vectors. These methods are compared to the state-of-the-art in terms of the training model (underlying optimization problem) and computational complexity. Section 5 shows the performance of the methods on two applications: intrinsic and extrinsic tasks in natural language processing, and regression tasks based on protein sequences in bioinformatics. Conclusions are given in Section 6.

2. Preliminaries: natural language processing and word representations

Word embeddings naturally intervene in Natural Language Processing (NLP) tasks, where many sub-problems (e.g. text classification, machine translation, named entity recognition) rely on vector representations of words and sentences [8]. They are also useful in bioinformatics, since proteins can be seen as a sequence of k-mers, for which vector representations can be used for regression tasks, for example, in channelrhodopsin localization, or thermostability prediction [9]. For both applications, we refer to a *corpus* as a set of documents, where each document is a sequence of tokens that are symbols from the vocabulary $\mathcal{V}$. We refer to such symbols as “words”: language words in natural language, or k-mers (short amino acid sequences) in protein sequences [9]. In this work, we focus on static embeddings, namely, in methods for determining vector representions of words, called *word vectors*, from co-occurence data.

In general, the construction of these static word vectors can be cast as the following optimization problem:

$$\min_{v} \quad \sum_{i} \sum_{j} \ell(v_i, v_j, C_{ij}), \tag{1}$$

where $v : \mathcal{V} \to \mathbb{R}^K$, $v_i := v(i)$ and ℓ is a loss function which depends on the vectors v_i, v_j and the co-occurrence C_{ij}: the number of times words i and j occur in the same *window* of w consecutive tokens in the corpus (by convention, our context windows do not overlap between document boundaries). Since the objective function of (1) is the sum of a large number of terms, usually Stochastic Gradient Descent (SGD) methods [10, 11] are used to solve this problem.

It is interesting to notice that in many word embedding methods that can be described by formulation (1), the loss function ℓ depends either on the Euclidean distance or on the inner product between the vectors v_i and v_j. This motivates us to consider the construction of word representations based on Euclidean Distance Geometry, where the fundamental problem consists in identifying point positions from information about a subset of their pairwise distances and/or inner products [7].

The DG literature provides several tools to address this problem in many situations. When all pairwise distances and/or inner products are available and exact, a solution in dimension K can be found from the top K eigenpairs of the corresponding Gramian [6]. Even when some pairwise information is missing, it is possible to solve the distance geometry problem in linear time, in the size of the vocabulary, using a Geometric Build-up algorithm [12].

In the next section we review key results from the theory of Distance Geometry and discuss two algorithms that will be used latter to build word vector representations. Even though these theoretical results are well-known, we decided to provide proofs for those results that directly related to the algorithms we shall discuss.

3. Distance geometry

Distance Geometry is the study of geometry based on the concept of distance rather than points and lines. For a deep and comprehensive discussion on this subject and its applications we recommend the survey [7] and references therein.

In this paper we focus on methods for solving the main problem in DG: given a partial set of distances between entities, find positions for these entities in some Euclidean space of dimension K so that they are compatible with the given distances. A partial set of distances can be represented by a weighted graph where the vertices $i \in \mathcal{V}$ represent the entities and an edge $\{i, j\}$ with weight d_{ij} belongs to the edge set $\mathcal{E}$ if the distance d_{ij} between i and j is known.

Definition 1 (Distance Geometry Problem)**.** *Given an integer $K > 0$ and a simple, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and an edge weight function $d : \mathcal{E} \to \mathbb{R}_+$, determine whether there exists a realization $v : \mathcal{V} \to \mathbb{R}^K$, such that*

$$\forall \{i, j\} \in \mathcal{E}, \quad \|v(i) - v(j)\| = d(\{i, j\}). \tag{2}$$

In (2), $\|.\|$ denotes the Euclidean norm. From Definition 1, we see that a DGP *instance* is given by the triplet $(\mathcal{G}, d, K)$. From here on, we shall use the compact notation $v_i := v(i)$, for all $i \in \mathcal{V}$, and $d_{ij} := d(\{i, j\})$, for all $\{i, j\} \in \mathcal{E}$, so that we can write (2) as

$$\forall \{i, j\} \in \mathcal{E}, \quad \|v_i - v_j\|^2 = d_{ij}^2. \tag{3}$$

The reason for squaring the distances will be clear soon. Let us denote the number of vertices by $n = |\mathcal{V}|$, and associate each element in $\mathcal{V}$ with an integer in $\{1, \ldots, n\}$. A sequence of vectors $v_1, \ldots, v_n \in \mathbb{R}^K$ satisfying (3) is called a *valid realization*. We remark that a realization can also be represented by a short, fat matrix $V \in \mathbb{R}^{K \times n}$, the i-th column of which is the vector corresponding to vertex $i \in \mathcal{V}$.

Although DGP is known to be NP-Hard [13], some special cases can be solved in polytime. The most relevant polynomial time case is that of complete graphs, which corresponds to a fully defined *Distance Matrix* $D = (d_{ij}^2)$.

Definition 2. *We say that a symmetric matrix $D \in \mathbb{R}^{n \times n}$, with null diagonal and non-negative entries is an* Euclidean *Distance Matrix (EDM) when* (3) *admits a solution for some dimension $K \leq n - 1$. The smallest positive integer K for which* (3) *has a solution is called* embedding dimension.

In this case of a complete graph $\mathcal{G}$ we can solve (3) or determine its infeasibility by a process similar to Classic Multidimensional Scaling (MDS) and Principal Components Analysis (PCA) [14], as detailed in the following.

Let $\mathbf{1}$ denote the vector of ones of appropriate dimension and $\langle \cdot, \cdot \rangle$ the standard inner product in $\mathbb{R}^K$. For a square matrix Z, $\text{diag}(Z)$ denotes a column vector containing the diagonal elements of Z. Let $\{v_i\}_{i=1}^n \subset \mathbb{R}^K$ be a set of points and $D \in \mathbb{R}^{n \times n}$ the corresponding EDM such that $D_{ij} = \|v_i - v_j\|^2$. Define the Gramian $G \in \mathbb{R}^{n \times n}$ such that $G_{ij} = \langle v_i, v_j \rangle$. Without loss of generality, assume that the points are centered around the origin: $\sum_i v_i = 0$. Then, from the relation between the inner product and Euclidean norm:

$$\|v_i - v_j\|^2 = -2\langle v_i, v_j \rangle + \|v_i\|^2 + \|v_j\|^2, \tag{4}$$

one can show that

$$D = \mathcal{K}(G) := -2G + \mathbf{1}\mathrm{diag}(G)^\top + \mathrm{diag}(G)\mathbf{1}^\top. \tag{5}$$

The linear map $\mathcal{K}$ when restricted from the space of symmetric centered matrices $\mathbb{S}_C = \{Y \in \mathbb{R}^{n\times n} \ :\ Y = Y^\top,\ Y\mathbf{1} = 0\}$ to the space of symmetric null diagonal matrices $\mathbb{S}_H = \{Z \in \mathbb{R}^{n\times n} \ :\ Z = Z^\top,\ \mathrm{diag}(Z) = 0\}$, is an isomorphism [15], whose inverse is given by

$$\mathcal{K}^{-1}(D) = -\frac{1}{2}JDJ \tag{6}$$

where $J = \mathrm{I}_n - (1/n)\mathbf{1}\mathbf{1}^\top$ is known as centering matrix (I_n denotes the identity matrix of order n).

Due to this one-to-one correspondence, we have the following equivalence.

Proposition 1. *Let $(\mathcal{G}, d, K)$ be a DGP instance where $\mathcal{G}$ is a complete graph, $D = (d_{ij}^2)$ the corresponding Distance Matrix, $G = \mathcal{K}^{-1}(D)$ and $v : \mathcal{V} \to \mathbb{R}^K$ such that $\sum_i v_i = 0$. Then, v solves* (3) *if, and only if, it solves*

$$\forall\ \{i,j\},\ \langle v_i, v_j\rangle = G_{ij}. \tag{7}$$

Proof. Let v be a solution of the DGP instance $(\mathcal{G}, d, K)$, such that $\sum_i v_i = 0$. Since

$$G = -\frac{1}{2}JDJ = -\frac{1}{2}\left(D - \frac{1}{n}\mathbf{1}(\mathbf{1}^\top D) - \frac{1}{n}(D\mathbf{1})\mathbf{1}^\top + \frac{1}{n^2}(\mathbf{1}^\top D\mathbf{1})\mathbf{1}\mathbf{1}^\top\right),$$

it follows that

$$\begin{aligned} G_{ij} &= -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_{k=1}^n d_{kj}^2 - \frac{1}{n}\sum_{k=1}^n d_{ik}^2 + \frac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n d_{ij}^2\right) \\ &= -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_{k=1}^n \|v_k - v_j\|^2 - \frac{1}{n}\sum_{k=1}^n \|v_i - v_k\|^2 + \frac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n \|v_i - v_j\|^2\right). \end{aligned}$$

Now, by using $\sum_{k=1}^n \|v_k - v_j\|^2 = \sum_{k=1}^n \|v_k\|^2 + n\|v_j\|^2$, we obtain

$$G_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \|v_i\|^2 - \|v_j\|^2\right) \overset{(4)}{=} \langle v_i, v_j\rangle.$$

Conversely,

$$D_{ij} = (\mathcal{K}(G))_{ij} = -2G_{ij} + G_{ii} + G_{jj} = -2\langle v_i, v_j\rangle + \|v_i\|^2 + \|v_j\|^2 = \|v_i - v_j\|^2.$$

□

A remarkable result in DG is Schoenberg's theorem [16, 17, 6], which states necessary and sufficient conditions for a Distance matrix to be Euclidean.

Theorem 1. *A Distance Matrix $D = (d_{ij}^2)$ is Euclidean if and only if $G = (-1/2)JDJ$ is positive semidefinite. Moreover, the embedding dimension is given by $r = \mathit{rank}(G)$.*

We remark that if $G = (-1/2)JDJ$ is positive semidefinite (PSD), then it is a genuine Gram matrix, and therefore it follows the existence of $V \in \mathbb{R}^{r \times n}$ such that $G = V^\top V$, where $r = \text{rank}(G)$. Then, from Proposition 1, the columns of V are a solution for (3) in dimension r.

The matrix V can be obtained as follows. Let $G = \sum_{i=1}^{n} \lambda_i u_i u_i^\top = U \Lambda U^\top$ be the spectral decomposition of G such that Λ is a diagonal matrix containing the eigenvalues of G in non-increasing order. The columns of U contain the corresponding eigenvectors. Then, $V = \sqrt{\Lambda_r} U_r^\top$, where Λ_r is a $r \times r$ diagonal matrix with the top r eigenvalues of G, and the columns of U_r contain the corresponding eigenvectors.

Concerning the related DGP defined by $(\mathcal{G}, d, K)$ (with $\mathcal{G}$ being complete), if $K \geq r$, the columns of V give a valid realization. Otherwise, when $K < r$, the corresponding DGP instance has no solution.

Even when the embedding dimension r of an EDM D is greater than the target dimension K, one may still want a representation of the underlying point set in a lower dimensional space, such that the interpoint inner products are preserved as most as possible. For this, we can choose among the PSD matrices Y of rank at most K, one that minimizes $\|Y - G\|_F$. A solution is given by $U_K \Lambda_K U_K^\top$, where Λ_K is diagonal with top K eigenvalues of G and $U_K \in \mathbb{R}^{K \times n}$ contains the corresponding eigenvectors in its columns [18, 19]. Thus, we call $V = \sqrt{\Lambda_K} U_K^\top$ an *approximate* realization for (3), in dimension K.

Last, but not least, if D is not an EDM (e.g. because some d_{ij} comes from a noisy measurement), then $G = (-1/2)JDJ$ is not PSD. Still, a solution in the least-squares sense is provided by

$$V^+ = \sqrt{\Lambda_K^+} U_K^\top, \tag{8}$$

where $\Lambda_K^+ = \max(\Lambda_K, 0)$ and the $\max(\cdot,\cdot)$ is componentwise.

Proposition 2. *Let $G \in \mathbb{R}^{n \times n}$ be a symmetric matrix. A positive semidefinite matrix with rank at most K that minimizes $\|Y - G\|_F^2$ is given by $Y^+ = U_K \Lambda_K^+ U_K^\top$.*

Proof. Since G is symmetric, it admits a spectral decomposition $G = U \Lambda U^\top$ where U is unitary and Λ is diagonal, containing the eigenvalues of G in non-increasing order $\lambda_1 \geq \cdots \geq \lambda_p > 0 \geq \lambda_{p+1} \geq \ldots \lambda_n$.

Due to the Frobenius norm invariance under unitary transformations

$$\begin{aligned} \|Y - G\|_F^2 = \|U^\top Y U - \Lambda\|_F^2 &= \sum_{i=1}^{n} \left[(U^\top Y U)_{ii} - \Lambda_{ii}\right]^2 + \sum_{i \neq j} (U^\top Y U)_{ij}^2 \\ &\geq \sum_{i=1}^{n} \left[(U^\top Y U)_{ii} - \Lambda_{ii}\right]^2 . \end{aligned}$$

By choosing $Y = U \Sigma U^\top$, where $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$, the above inequality holds as equality and

$$\|Y - G\|_F^2 = \sum_{i=1}^{p} (\sigma_i - \lambda_i)^2 + \sum_{i=p+1}^{n} (\sigma_i - \lambda_i)^2.$$

The constraint that Y must be positive semidefinite forces $\sigma_i = 0$, $i = p+1, \ldots, n$, in order to minimize the second term in the sum above. In case $K < p$, we also need to set $\sigma_i = 0$, for $i = K+1, \ldots, p$ to ensure that the rank of Y is at most K, and $\sigma_i = \lambda_i$, $i = 1, \ldots, K$ for minimizing the first term. Therefore

$$\begin{aligned} Y^+ &= U \operatorname{diag}(\max\{\lambda_1, 0\}, \ldots, \max\{\lambda_K, 0\}, 0, \ldots, 0)\, U^\top \\ &= U_K \operatorname{diag}(\max\{\lambda_1, 0\}, \ldots, \max\{\lambda_K, 0\})\, U_K^\top = U_K \Lambda_K^+ U_K^\top. \end{aligned}$$

□

We remark that the approximate realization given by (8) is a solution of the following optimization problem

$$\min_{V \in \mathbb{R}^{K \times n}} \quad \|V^\top V - G\|_F^2 = \sum_i \sum_j (\langle v_i, v_j \rangle - G_{ij})^2. \tag{9}$$

3.1. *Geometric Build-up*

If all pairwise distances are available, then we can solve (3) by computing an eigendecomposition in $O(n^3)$ operations. In fact, it is possible to do better than $O(n^3)$ under certain assumptions. Assume the graph $\mathcal{G}$ admits a *vertex order* ($<$) such that: (i) the first $m \geq K+1$ vertices form a clique; (ii) for all other vertices $i > m$, vertex i has at least $K+1$ *adjacent predecessors*. Then, when all available distances are exact, a solution to the corresponding DGP can be found in linear time $O(n)$ by a Geometric Build-up algorithm [12] that we shall describe ahead. We remark that such vertex orders, also known as $(K+1)$-lateration orders [20], can be found in polynomial time by a greedy algorithm [21].

For every vertex $i > m$, let $U(i) := \{j \in \mathcal{V} \;:\; \{j, i\} \in \mathcal{E} \text{ and } j < i\}$ be the set of all adjacent predecessors of i (with "$<$" being defined by the vertex order). Any subset $\delta(i)$ of $U(i)$, with cardinality $|\delta(i)| \geq K+1$ is called a set of *reference vertices* for vertex i.

The first m vertices can be realized using the process described in Section 3, leading to a cost bounded by $O(m^3)$. Assuming $|\delta(i)| = K+1$, for every $i > m$, then, following the vertex order, the position of every other vertex $i = m+1, \ldots, n$ can be found by solving the quadratic system:

$$\forall j \in \delta(i), \quad \|v_j - v_i\|^2 = d_{ji}^2 \tag{10}$$

where $\delta(i) = \{j_1, \ldots, j_{K+1}\}$ and $v_{j_1}, \ldots, v_{j_{K+1}}$ are the position vectors of $K+1$ (already placed/localized) adjacent predecessors of vertex i. It is not hard to show that when (10) admits a solution, it coincides with the one of a $K \times K$ linear system $Ax = b$, where A is nonsingular, provided $v_{j_1}, \ldots, v_{j_{K+1}}$ are affinely independent. See [12, 7] for further details.

This approach is known in the DG literature as Geometric Build-Up (GBU) [12, 22]. The total cost of GBU is given by $O(m^3) + (n-m)O(K^3)$, if the involved matrices show no special structure. In the following, we discuss how this complexity can be further improved when the set of references $\delta(i)$ is fixed and how to used inner products instead of distances. These requirements will meet our application of determining word representations from co-occurrence data in Section 4.

3.2. GBU with inner products and fixed references

If all pairwise distances between vertices in $\delta(i) \cup \{i\} = \{j_1, \ldots, j_M, i\}$, with $M \geq K + 1$, are known, then, due to Proposition 1, the system in (10) is equivalent to

$$\forall j \in \delta(i), \quad \langle v_j, v_i \rangle = G_{ij} \tag{11}$$

where G_{ij} denotes an entry of the Gram matrix G which, if not directly available, can be computed from the linear isomorphism between distance and Gram matrices: by applying (6) to a submatrix of D containing the squared distances corresponding to the subset of vertices $\delta(i) \cup \{i\}$. Notice that (11) is also a linear system of the form $Ax = b$, with $A^\top = (v_{j_1} \ \ldots \ v_{j_M}) \in \mathbb{R}^{K \times M}$ and $b = (G_{j_1,i}, \ldots, G_{j_M,i})^\top$. Let us suppose that the reference vertices for every vertex $i > m \geq K + 1$ are

Algorithm 1 Geometric Build-up with fixed references

Input: (Pseudo-) Gramian $G \in \mathbb{R}^{n \times n}$, integer K (dimension), integer m (number of references)

Output An approximate realization matrix $V \in \mathbb{R}^{K \times n}$

```
1:  $G_0 \leftarrow G(1:m, 1:m)$
2:  $V \leftarrow \mathbf{0} \in \mathbb{R}^{K \times n}$
3:  Compute the top $K$ eigenvalues (and eigenvectors) of $G_0$: $\Lambda_K, U_K$
4:  Set $A^\top = (v_1 \ldots v_m) = \sqrt{\Lambda_K^+} U_K^\top$ and $V(:, 1:m) = A^\top$
5:  Compute QR decomposition: $A = QR$
6:  for $i = m+1, \ldots, n$ do
7:      $b_i = G(1:m, i)$
8:      Solve $Rv_i = Q^\top b_i$
9:      $V(:, i) = v_i$
10: end for
11: return $V$
```

fixed as $\delta(i) = \{1, \ldots, K+1, \ldots, m\}$, which implies $|\delta(i)| = M = m$, for each $i > m$. In this case, in the linear system of (11), although the right hand side vector $b = (G_{1,i}, \ldots, G_{m,i})^\top$ changes for each i, the coefficient matrix $A^\top = (v_1 \ \ldots \ v_m)$ is the same for every $i > m$. Thus, concerning the solution of linear systems $Av_i = b_i$, for $i = m+1, \ldots, n$, we can factor the matrix $A \in \mathbb{R}^{m \times K}$ *only once* and exploit its factorization to actually solve triangular systems of order K for each $i > m$.

Since the measurements G_{ij} can be noisy, in fact we consider a least-squares solution for the aforementioned linear systems. Recall that the least-squares solution of an overdetermined, full-rank system of linear equations $Ax = b$, i.e x that minimizes $\|Ax - b\|^2$, is given by the solution of the triangular system $Rx = Q^\top b$, where $A = QR$, with $R \in \mathbb{R}^{K \times K}$ and $Q \in \mathbb{R}^{m \times K}$ is the "economy size" QR decomposition of A [23].

This scheme leads to a cost of $O(m^3) + (n-m)O(K^2)$, where G_0, the first submatrix of G of order m is realized using spectral decomposition (see Section 3), followed by QR decomposition of $A = (v_1 \ \ldots \ v_m)^\top$, and the positions of the remaining $n - m$ vertices are found by solving the triangular systems $Rv_i = Q^\top b_i$, for $i = m+1, \ldots, n$. Since $A = U_K \sqrt{\Lambda_K^+}$, where the columns of U_K are the eigenvectors corresponding to the top K eigenvalues of G_0, Av_i may be interpreted as the projection of $b_i = (G_{1,i}, \dots, G_{m,i})^\top$ onto the eigenspace spanned by these eigenvectors.

These ideas are summarized in Algorithm 1. We have used the notation $A(:, j)$ and $A(i, :)$ for the j-th column and i-th row of a matrix A, respectively, and $1 : m$ means that an integer index varies from 1 to m. We remark that whenever the values G_{ij} are exact and the affine dimension of $v_1, \dots, v_m$ is K, the above procedure yields an exact solution to (3) [24].

3.3. Vertex order and loss function

The DGP we aim to solve with this particular version of GBU can also be cast as the following unconstrained optimization problem:

$$\min_{V \in \mathbb{R}^{K \times n}} \quad \sum_{i=1}^{m} \sum_{j=i}^{m} (\langle v_i, v_j \rangle - G_{ij})^2 + \mathcal{L}(V, G) := g(V), \tag{12}$$

where $\mathcal{L}(V, G) = \sum_{i=m+1}^{n} \sum_{j \in \delta(i)} (\langle v_i, v_j \rangle - G_{ij})^2$ and $m \geq K + 1$ is the size of the initial clique (number of references/anchors). We remark that, when $|\mathcal{E}| = n(n-1)/2$ and $\delta(i) = U(i)$, problem (12) is equivalent to (9).

Usually DGP graphs arising from many problems are quite sparse in practice. Even if they admit a $(K+1)$-lateration order, the set of reference vertices $\delta(i)$ usually changes for each $i > m \geq K + 1$. However, when the underlying graph $\mathcal{G}$ is complete, *any* vertex order is in fact a $(K + 1)$-lateration order. We highlight that this will be our case, since we know all entries of the co-occurrence (or empirical PMI) matrix (see Section 4), from which we obtain the adjacency information. Thus, we shall be able to apply GBU with fixed references, where $\delta(i) = \{1, \dots, m\}$, for every $i > m \geq K + 1$.

Therefore, in the GBU-based methods discussed in this paper, the proposed vertex (word) orders are simply aimed at improving the quality of the word vectors. These orders will determine which entries of the Gram matrix G ($\approx$ PMI matrix) are taken into account in the GBU method. Thus, in the objective function of (12), the vertex order determines the weight of the terms $(\langle v_i, v_j \rangle - G_{ij})^2$: 1 for edges $\{i, j\}$ used in the sequential build-up process and 0 for the others.

In the problem of determining word vector representations, one possible vertex (word) order is to consider words in decreasing order of their frequency in the corpus. Another candidate for vertex ordering can be *coreness*. We remind the reader that the k-core of a graph is the maximum cardinality subgraph such that each vertex has an induced degree at least k. Then, the *coreness* of a vertex is defined as the maximum integer k such that it belongs to a k-core and not to a $(k + 1)$-core. Computing k-cores can be done in linear time [25]. High coreness intuitively corresponds to words belonging to a dense community of frequent words, hence suggesting a natural vertex order both for GBU and the Divide and Conquer (DC) algorithm presented in the next subsection. In the experiments presented in Section 5, the results using word frequencies or coreness were very similar.

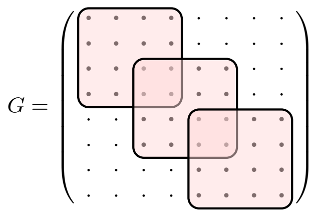

Figure 1: Example for the divide step ($|I_1| = |I_2| = 2$, $n = 8$, $K = 1$).

3.4. Divide and conquer (DC)

Given a candidate Gram matrix $G \in \mathbb{R}^{n\times n}$, another approach for solving the corresponding DGP is to split the problem in P subproblems, solve each subproblem by spectral decomposition (as described in Section 3) and merge the partial solutions together by solving a sequence of Procrustes problems [26].

This approach is called Divide and Conquer (DC) and consists in the two following steps:

Divide: consider submatrices G_i (for $i \leq P$) of G, each having size $n_i \times n_i$, such that the following conditions hold. (i) Each G_i is centered along the diagonal; (ii) G_i and G_{i+1} have at least $K+1$ points in common indexed by I_i; (iii) $\sum_{i=1}^{P-1}(n_i - |I_i|) + n_P = n$. The division is illustrated in Figure 1. Each submatrix defines a DGP sub-instance which is solved with a method such as matrix factorization or GBU to realize the corresponding points.

Conquer (Merge): after the solution of each sub-instance, we have to combine the partial realizations consistently in order to obtain a realization of the whole graph. This operation is carried out sequentially as follows. The solution of the first sub-instance is saved. Then, for an instance $i+1$, for $i \geq 1$, the number I of common points between sub-instances i and $i+1$ must be at least $K+1$ in order to define unique translations and rotations for the common points to be aligned. Let X_{i+1} be the current solution obtained in the divide step, $V_i \in \mathbb{R}^{K\times n_i}$ be aligned vectors obtained at the previous step i, and let $A(:,j)$ denote the j-th column of a matrix A. Then X_{i+1} can be aligned by using Procrustes analysis [26]: the best alignment rotation $\hat{Q}_i$ and translation $\hat{T}_i$ are

$$\hat{Q}_i, \hat{T}_i = \underset{Q\in O_K, T\in\mathbb{R}^K}{\operatorname{argmin}} \sum_{k=1}^{I} \|V_i(:, n_i - I + k) - (QX_{i+1}(:,k) + T)\|_2^2,$$

where O_K is the set of orthogonal matrices of order K. We recall that the solution for this orthogonal Procrustes problem, assuming $\sum_{k=1}^{I} X_{i+1}(:,k) = 0$, is given by $\hat{T}_i = (1/I)\sum_{k=1}^{I} V_i(:, n_i - I + k)$ and $\hat{Q}_i = \hat{U}\hat{V}^\top$, where $V_c X_{i+1}(:, 1:I)^\top = \hat{U}\hat{\Sigma}\hat{V}^\top$ is the Singular Value Decomposition (SVD) of the $K \times K$ matrix $V_c X_{i+1}(:, 1:I)^\top$, with $V_c = V_i(:, (n_i - I + 1) : n_i) - \hat{T}_i \mathbf{1}^\top$. The aligned vectors are then given by $V_{i+1} = \hat{Q}_i X_{i+1} + \hat{T}_i\,\mathbf{1}^\top$.

The pseudo-code for DC is presented in Algorithm 2. Although the computational cost of DC may be higher than the cost of GBU with fixed references, the DC approach may take into account more terms G_{ij} than GBU does and still be less expensive than

Algorithm 2 Divide and conquer

Input: (Pseudo-) Gramian $G \in \mathbb{R}^{n \times n}$, integer K (dimension), subproblem sizes $n_1, \ldots, n_P$.

Output An approximate realization matrix $V \in \mathbb{R}^{K \times n}$

```
 1: $G_1 = G(1 : n_1, 1 : n_1)$
 2: Compute the top $K$ eigenvalues (and eigenvectors) of $G_1$: $\Lambda_K, U_K$
 3: $V \leftarrow \mathbf{0} \in \mathbb{R}^{K \times n}$
 4: Set $V(:, 1 : n_1) = V_1 = \sqrt{\Lambda_K^+} U_K^\top$, $\ell = n_1$
 5: for $i = 2, \ldots, P$ do
 6:     $\hat{I}_i = \{\ell - K, \ldots, \ell - K + n_i - 1\}$
 7:     $G_i = G(\hat{I}_i, \hat{I}_i)$
 8:     Compute the top $K$ eigenvalues (and eigenvectors) of $G_i$: $\Lambda_K, U_K$
 9:     Set $X_i = \sqrt{\Lambda_K^+ U_K^\top}$
10:     Align common points of $G_{i-1}$ and $G_i$ with Procrustes analysis: $\hat{Q}_{i-1}, \hat{T}_{i-1}$
11:     $V_i = \hat{Q}_{i-1} X_i + \hat{T}_{i-1} \mathbf{1}^\top$
12:     $V(:, \hat{I}_i) = V_i$
13:     $\ell \leftarrow \ell - K + n_i - 1$
14: end for
15: return $V$
```

the spectral decomposition of the whole Gramian G (see Table 3 and the discussion in the Subsection 4.2). Besides, the Divide and Conquer method is more flexible than GBU with fixed references, since it is not mandatory to use the first m vertices as references. Also, it allows to increase the size of sub-instances, taking in account more inner product terms of the objective function of (9).

3.5. *Stochastic Gradient Descent for DGP*

In Section 2 we mentioned that most of the standard methods for static word representations solve an optimization problem similar to (1) by employing SGD (Stochastic Gradient Descent). It turns out that problem (12) is a particular case of (1) and thus it raises the question of whether it is advantageous to use DG-based methods instead of simply applying SGD to (12).

To answer this question, we report some numerical experiments on synthetic datasets to illustrate the difference in performance between GBU and SGD.

We generate random datasets varying the number of points n, the dimension K and the noise level η, following the procedure described in Algorithm 3 (see Appendix A). After generating n random points $V_1, \ldots V_n$ in $\mathbb{R}^K$, corresponding to columns of a matrix V, we build the true Gramian $G^{\mathsf{true}} = V^\top V$ and perturb its entries G^{true}_{ij} by a random noise $\epsilon_{ij} \sim \mathcal{N}(0, \eta \times G^{\mathsf{true}}_{ij})$ to obtain $G_{ij} = G^{\mathsf{true}}_{ij} + \epsilon_{ij}$.

The optimization formulation we consider, which is equivalent to (12), is:

$$\min_{V \in \mathbb{R}^{K \times n}} \quad \frac{1}{N} \left[\sum_{i=1}^{m} \sum_{j=i}^{m} (\langle v_i, v_j \rangle - G_{ij})^2 + \mathcal{L}(V, G) \right], \tag{13}$$

Table 1: GBU vs. SGD to solve Problem 13 with $m = K + 1$

	η	Method	Performance ($K = 3$)		Performance ($K = 50$)		Performance ($K = 200$)	
			Score	Time	Score	Time	Score	Time
$n = 500$	0	GBU	**4.480e-32**	**1.930e-02**	**1.931e-28**	**4.652e-02**	**7.437e-29**	**1.20e-01**
		SGD	1.224e-03	1.485e+01	1.009e-02	1.593e+01	1.266e-02	1.719e+01
	0.01	GBU	**7.181e-07**	**1.950e-02**	**9.932e-06**	**2.149e-02**	*	*
		SGD	1.500e-03	1.394e+01	1.069e-02	1.595e+01	**1.306e-02**	**1.706e+01**
	0.02	GBU	**3.762e-06**	**2.013e-02**	**6.148e-06**	**4.089e-03**	*	*
		SGD	1.182e-03	1.364e+01	1.055e-02	1.599e+01	**1.416e-02**	**1.705e+01**
$n = 1000$	0	GBU	**1.794e-32**	**4.598e-02**	**5.105e-30**	**9.387e-02**	**6.757e-29**	**2.99e-01**
		SGD	1.121e-03	4.596e+01	1.335e-02	5.543e+01	2.680e-02	5.943e+01
	0.01	GBU	**3.360e-07**	**4.568e-02**	**4.960e-06**	**1.898e-02**	*	*
		SGD	1.167e-03	5.054e+01	1.367e-02	5.600e+01	**2.743e-02**	**5.990e+01**
	0.02	GBU	**1.757e-06**	**4.809e-02**	*	*	*	*
		SGD	1.434e-03	5.189e+01	**1.336e-02**	**5.774e+01**	**2.892e-02**	**5.939e+01**

where $N = m(m+1)/2 + m(n-m)$, $\mathcal{L}(V,G) = \sum_{i=m+1}^{n} \sum_{j\in\delta(i)} (\langle v_i, v_j\rangle - G_{ij})^2$ and $m \geq K+1$ is the size of the initial clique.

We used ADAM procedure for the stochastic gradient implementation, available from Pytorch [27]. We set a maximum number of epochs to 1000 or stop when the objective function reaches a value smaller than 10^{-3}.

First, we consider problem (13) with $m = K + 1$ which is also the (minimum) number of references in GBU. Table 1 presents the performance of GBU and SGD for $n = 500, 1000$, $K = 3, 50, 200$ and $\eta = 0, 0.01, 0.02$ in terms of running time (in seconds) and final value of the loss function (score). These figures are averages over 10 runs for each triplet (n, K, η). A symbol "*" means that a method fails due to numerical instability. As one can see, GBU systematically performs better than SGD when the data are exact, both in time and score. However, GBU ran into numerical problems for $K = 200$ on instances with noise. We observed that the matrix R, from the QR decomposition of A (see Section 3.2), happens to be near singular in these cases. A possible explanation is that the volume of the convex hull of the $m = K + 1$ reconstructed points $v_1, \ldots, v_m$ in $\mathbb{R}^K$ is close to zero. Thus, the main advantage of SGD over GBU for these problems (with $m = K + 1$) is its resistance to noise and numerical stability.

Nevertheless, by increasing the number of references m, it is more likely that the volume of the convex hull of $v_1, \ldots, v_m$ is sufficiently positive or, in other words, that the matrix R is not so ill-conditioned. This claim is corroborated by the experiments in Table 2 where we increased a bit more the number of references m (to $1.5 \times K$ instead of $K + 1$). We can see that GBU becomes stable, and outperforms SGD in all instances with noise level $\eta = 0.02$. This empirical analysis partially explains our choice $m = 4K$ for GBU in the experiments of Section 5.

Further discussion comparing the computational complexity of SGD and GBU is provided in Appendix B.

Table 2: GBU vs. SGD to solve Problem 13 with $m = \lceil 1.5K \rceil$

	η	Method	Performance ($K = 3$)		Performance ($K = 50$)		Performance ($K = 200$)	
			Score	Time	Score	Time	Score	Time
$n = 500$	0.02	GBU	**3.617e-06**	**1.948e-02**	**5.275e-04**	**5.246e-02**	**5.609e-03**	**1.21e-01**
		SGD	1.350e-03	1.470e+01	1.230e-02	1.592e+01	1.621e-02	1.711e+01
$n = 1000$	0.02	GBU	**2.203e-06**	**4.591e-02**	**5.587e-04**	**1.070e-01**	**7.524e-03**	**3.67e-01**
		SGD	1.061e-03	4.541e+01	1.646e-02	5.647e+01	3.525e-02	6.039e+01

4. Distance Geometry for Word Representations

In this section, we explain how the main concepts and methods of Distance Geometry can be used to determine word vector representations from co-occurrence data extracted from a text corpus.

A natural question is whether there exists a pertinent distance between words of the vocabulary. For instance, in the case of natural language, one wishes to consider a distance between words that measures their "semantic difference". One could expect that such a distance, even if only partially defined, would yield a set of word vectors satisfying the property: (A) two words are semantically correlated if their corresponding vectors are close. Here, semantic correlation can be interpreted loosely (e.g synonymy, antonym, or more complicated forms of semantic correlation). However, a function verifying the property (A) may not satisfy the distance axioms. Furthermore, as discussed in [28], co-occurrence rates also do not satisfy metric constraints. Then, according to [29], it is more reasonable to consider the statistical nature of the co-occurrence data, and to interpret observed object (word) pairs i and j as drawn from a joint distribution that is determined by distances or inner products between vectors of the underlying low-dimensional embedding.

Let $p(i, j)$ be the probability of finding words i and j in the same window (a sequence of w consecutive tokens in a corpus), and $p(i)$, $p(j)$ denote the marginal probabilities. The Pointwise Mutual Information (PMI):

$$\mathsf{PMI}(i, j) := \log \frac{p(i, j)}{p(i)p(j)}$$

is an information theoretic measure that can be used to model associations between words [30] and is widely used in count-based and matrix factorization based word embeddings [31]. In this work, we shall focus on word vectors that approximately solve the following optimization problem:

$$\min_{v_i, v_j \in \mathbb{R}^K} \quad \sum_i \sum_j \left(\langle v_i, v_j \rangle - \mathsf{PMI}(i, j) \right)^2 . \tag{14}$$

The reasoning behind (14) is that if the "similarity" between vectors v_i and v_j is measured by their inner product $\langle v_i, v_j \rangle$, then two words with a high degree of association (high PMI), should have a high value of $\langle v_i, v_j \rangle$, and vice-versa. Similar models based on cossine-similarity were previously used in the literature [32].

Comparing (14) with (9) (or with (12)) suggests to consider $G_{ij} \approx \mathsf{PMI}(i,j)$. However, recent works [33, 34] have shown that the symmetric PMI matrix obtained from word-word co-occurrences fails to be positive semidefinite. Thus, we do not expect the PMI matrix to be a Gram matrix, i.e, the optimal value of (14) to be zero. Nevertheless, we observe that the solution of (14) provides the best rank-K positive semidefinite approximation for the PMI matrix (see Proposition 2). Furthermore, there is empirical evidence in the literature [33, 34] that the K largest eigenvalues of PMI (for $K \leq 1000$, at least) are positive.

Therefore, our approach to determine word vectors consists in applying the Distance Geometry methods of Section 3.2 (GBU) and Section 3.4 (DC) to approximately solve the DGP expressed by (7) using $\mathsf{PMI}(i,j)$ in place of G_{ij}.

4.1. Co-occurrences and PMI estimator

Our input will be the word-word co-occurrence matrix C. Following [31], we use an information theoretic measure, the pointwise mutual information $\mathsf{PMI}(i,j) = \log(p(i,j)/(p(i)p(j)))$ as a measure of association between words [30].

Actually, since the true probabilities are unknown, we consider an empirical PMI matrix G^0 whose entries are $G^0_{ij} = \log \rho_{ij}$, where

$$\rho_{ij} = \frac{C_{ij}}{\sum_k C_{kj} \sum_k C_{ik}} \sum_{k,l} C_{kl} \tag{15}$$

recalling that C_{ij} is the number of windows in which words i and j co-occurred.

However, notice that entries of G^0 corresponding to zero co-occurrences $C_{ij} = 0$ are not well defined. An alternative, commonly used in NLP [30, 31], is to consider the corrected empirical PMI matrix G, where

$$G_{ij} = \begin{cases} \log \rho_{ij}, & C_{ij} > 0 \\ 0, & C_{ij} = 0 \end{cases} \tag{16}$$

which is, moreover, a sparse matrix.

As we shall see in Section 4.2, many methods for word embeddings employ the empirical PMI between words i and j as a surrogate model for the inner product $\langle v_i, v_j \rangle$, at least implicitly [31, 35, 36].

4.2. Connections with other PMI-based methods and their complexities

In this section, we perform a theoretical comparison between the DG methods proposed in this paper and other popular word embedding methods. To assess similarities and differences between them, we analyze their underlying optimization problems and also provide the computational complexity of each method.

We recall that the empirical PMI matrix defined by (15) and (16) is used as a rough approximation of the Gram matrix G [31, 35].

PMI-eigs. Word vectors obtained from the Singular Value Decomposition (SVD) of an empirical word-context asymmetric PMI matrix have been used in NLP literature

[31]. For a symmetric word-word empirical PMI matrix, in view of (9), the word vectors are obtained from

$$\min_{V \in \mathbb{R}^{K \times n}} \quad \|V^\top V - G\|_F^2 = \sum_i \sum_j (\langle v_i, v_j \rangle - G_{ij})^2. \tag{17}$$

The solution of (17) is constructed from the top K eigenpairs of G as discussed in Section 3. We refer to this approach as "PMI-eigs". Due to the sparsity of $G \in \mathbb{R}^{n \times n}$, such eigenpairs are usually computed by an Implicitly Restarted Lanczos method (IRLM) [37] as implemented, for example, in the Matlab routine `eigs`. Its cost per iteration is given by $q\gamma n + (6K+9)qn + 4q^2n + 2K^2n + O((K+q)^3)$, where q is the number of shifts and γ is the average number of nonzero elements of rows of G. See [38] for details. If we denote $\tilde{m} = K + q$, then the cost can be written as $q\gamma n + nO(K^2) + O(\tilde{m}^3)$. Therefore, considering $\gamma q \ll K^2$, the computation of the top K eigenpairs of the PMI matrix requires $O((nK^2 + \tilde{m}^3)n_{\text{iter}})$ operations, where n_{iter} is the number of IRLM iterations.

GBU. By considering the GBU method of Section 3.2 with m fixed references, where $\delta(i) = \{1, \ldots, m\}$, for all $i > m$, we aim at solving the optimization problem (12). In this case, the objective function is similar to (17), but the sum is not over all pairs $\{i, j\}$, but only those implied by the vertex order. We consider a vertex order in which words are sorted in decreasing order of frequency in the corpus. Thus, according to the discussion in Section 3.2, word vectors $v_1, \ldots, v_m$ for the m most frequent words are obtained from the top K eigenpairs of the corresponding $m \times m$ Gramian G_0 whereas, for $i > m$, word vectors v_i come from projecting $b_i = (G_{1,i}, \ldots, G_{m,i})^\top$ onto the subpace spanned by such eigenvectors. As already mentioned in Section 3.2, the complexity of GBU with m fixed references is given by $O(m^3 + (n-m)K^2)$.

DC. The underlying objective function of DC is a variation of (17) but summing over the pairs $\{i, j\}$ corresponding to rows and columns of at least one of the submatrices $G_1, \ldots, G_P$ in the divide step. Let us consider exactly $K+1$ anchors (as in our experiments), i.e $|I_1| = ... = |I_{P-1}| = K+1$ and let $n'^{(i)}_{\text{iter}}$ represent the number of IRLM iterations to solve each sub-instance i of size n_i. Also, denote $\tilde{m}_i = K + q_i$, where q_i is the number of shifts in IRLM to solve a sub-instance. Then, the cost of DC can be estimated by $O(\sum_{i=1}^{P}(n_iK^2 + \tilde{m}_i^3)n'^{(i)}_{iter} + (K+1)^3(P-1))$, where the last term represents the cost of solving $P-1$ orthogonal Procrustes problems (merge step).

Notice that the above methods try to fit the inner products $\langle v_i, v_j \rangle$ to the empirical PMI G_{ij}. In [35], the relation between $\langle v_i, v_j \rangle$ and $\mathsf{PMI}(i, j)$ is studied based on a generative model, whereas in [36] the authors suggest that when the corpus size tends to infinity, for a window of size w sufficiently large, for each $\{i, j\}$, there exists a_i and b_i, such that $\|v_i - v_j\|^2 \approx -\log(C_{ij}) - a_i - b_j$. Both models claim to be consistent with matrix factorization methods [31] and others based on regression [2] under certain assumptions. In the following we mention three word embedding methods that fit into this class.

GloVe. In [2], the goal is to find embeddings $v, \tilde{v} : \mathcal{V} \to \mathbb{R}^K$, by solving a weighted least-squares regression problem

$$\min_{v,\tilde{v},b,\tilde{b}} \quad \sum_i \sum_j f(X_{ij}) \left(\langle v_i, \tilde{v}_j \rangle + b_i + \tilde{b}_j - \log(X_{ij}) \right)^2 \tag{18}$$

where X_{ij} is the number of times word j occurs in the context of word i, $f(X_{ij}) = \min(X_{ij}, 100)^{3/4}$. Therefore, if the "bias terms" b_i and $\tilde{b}_j$ were known, one could see (18) as a variant of (17) weighted by $f(X_{ij})$, by using $G_{ij} \approx \log(X_{ij}) - b_i - \tilde{b}_j$ and assuming $\tilde{v}_i = v_i$. Furthermore, if $b_i = \log(X_i/\sqrt{S})$ and $\tilde{b}_j = \log(X_j/\sqrt{S})$, where $X_i = \sum_j X_{ij}$ and $S = \sum_i \sum_j X_{ij}$, we have $G_{ij} \approx \log \rho_{ij}$, for $X_{ij} > 0$, i.e our empirical approximation for $\mathsf{PMI}(i, j)$ but using the counts X_{ij} rather than C_{ij}. We remark that whenever $X_{ij} = 0$, the corresponding term does not appear in (18), but in (12), it may contribute to the objective function if either the pair $\{j, i\}$ is in the initial clique or $j \in \delta(i)$.

word2vec & doc2vec. The skip-gram with negative sampling (SGNS) introduced in [1] aims at predicting the surrounding context words given a center word, using the maximum likelihood principle on the probability of the surrounding context words, given a word at position t. The objective of the skip-gram model is to maximize the empirical log-likelihood of the form $\sum_{t=1}^{p} \sum_{c \in C(t)} \log p(w_c|w_t)$, where $C(t)$ is the set of indices of words surrounding the word w_t, and p is the number of tokens (size of the training set). Therefore, using the binary logistic loss, the following negative log-likelihood is to be minimized:

$$\sum_{t=1}^{p} \left[\sum_{c \in C(t)} \left(h(\langle v_t, v_c \rangle) + \sum_{\eta \in N(t,c)} h(-\langle v_t, v_\eta \rangle) \right) \right] \tag{19}$$

where $h(x) := \log(1 + \exp(-x))$ and $N(t, c)$ is a set of negative examples (words that do not occur at context c of word t) sampled from the vocabulary. Usually, this optimization problem is solved approximately using Stochastic Gradient Descent [10, 11]. The doc2vec method also follows the same idea, except that each document is seen as a supplementary word for each context, and then attributed a vector as well. Therefore, these formulations are different from ours, or PMI-eigs and GloVe for that matter, but meet in the sense that they consider co-occurrences information as input (see [31] for a deeper discussion on the connection of (19) with the above optimization problems and implicit matrix factorization).

fastText. In [39], an extension of SGNS is proposed. This extension takes into account the morphology of words: a vector representation is associated to each character s-gram and words are represented as the sum of these vectors.

The complexity of the co-occurrence based methods discussed above are summed up in Table 3. Recall that $n = |\mathcal{V}|$ is the size of the vocabulary and K the dimension. Concerning DC and PMI-eigs, it should be noted that usually $n'_{\text{iter}} \ll n_{\text{iter}}$ hence PMI-eigs complexity is not necessarily lower than that of DC. The number of shifts q, q_i is

Table 3: Computational complexity for several word vectors realization algorithms

Method	**Complexity**
PMI-eigs	$O((nK^2 + \tilde{m}^3)n_{\text{iter}})$
GBU	$O(m^3 + (n-m)K^2)$
DC	$O(\sum_{i=1}^{P}(n_iK^2 + \tilde{m}_i^3)n'^{(i)}_{iter} + (K+1)^3(P-1))$
GloVe	$O(p^{0.8}n_{\text{epochs}}K)$

internally set in the implementation of `eigs` in Matlab and it is difficult to estimate, even though it is likely that $K \leq q \ll n$ and $K \leq q_i \ll n_i$. Besides, the number of iterations $n'_{\text{iter}}, n_{\text{iter}}$ depends on the distribution of the eigenvalues of the corresponding matrices. For these reasons, we do not know how to compare their theoretical complexity. However, we will compare their empirical running times in Section 5.

The complexity study for Glove is detailed in [2, Section 3.2] and indicates $O(p^{0.8}n_{\text{epochs}})$ where p is the total number of tokens in the corpus, and n_{epochs} the number of epochs of the stochastic gradient descent (SGD) [10, 11]. For a fair comparison with other methods, we consider that this complexity also depends linearly on the dimension. Since fastText, word2vec and doc2vec rely on SGD for solving the corresponding underlying optimization problems, we will assume that their complexity is similar to GloVe.

For the first three methods of Table 3, the dimension seems to be a drawback, but their complexities are good in practice. For example, for a corpus composed of $p = 2.66 \times 10^8$ tokens (corpus described in Section 5), $n_{\text{epochs}} = 15$ (standard corpus size and parameters for SGD), containing about $n = 10^5$ different words, dimension $K = 50$, we have $\mathcal{C}_{\mathsf{Glove}} = p^{0.8}n_{\text{epochs}}K \approx 4.13 \times 10^9$ and $\mathcal{C}_{\mathsf{GBU}} = m^3 + (n-m)K^2 \approx 2.50 \times 10^8$, with $m = 200$ references. These complexity estimates are consistent with running times in our experiments. It should be noted that these estimates do not take pre-processing of the corpus into account, which is $O(p)$ in all cases: this corresponds to one pass through the corpus in order to construct the vocabulary, and possibly ignore low-frequency terms.

5. Experiments

In order to assess the performance of word representations obtained by the proposed methods as input to machine learning algorithms for an end-task, as well as to compare training times with well established methods in the literature, we consider two sets of experiments: (i) protein sequence embeddings in bioinformatics and (ii) natural language processing. The first set was chosen to illustrate the usefulness of our methods in applications where pre-trained representations are rarely available whereas the second aims to contrast the training time with well-known methods for word representation in the NLP literature.

Table 4: Results for Channelrhodopsin localization (Localization) and Thermostability (T50) regression tasks. All embeddings have dimension $K = 64$. First and second best are in bold and underline, respectively.

	Localization		T50	
Representation	R^2	MAE	R^2	MAE
doc2vec ($k = 3$, $w = 3$)	0.52	0.756	0.35	3.02
doc2vec ($k = 3$, $w = 7$)	0.47	0.787	<u>0.45</u>	<u>2.94</u>
GBU ($k = 3, w = 3$)	<u>0.54</u>	<u>0.728</u>	0.42	3.00
GBU ($k = 3, w = 7$)	**0.61**	**0.667**	**0.48**	**2.69**

The codes and data are available in our repository[1]. All experiments were carried out in a personal laptop with CPU of 2 cores Intel(R) Core i5 1.8Ghz with 8 Gb of Ram.

5.1. Protein sequence embeddings

An interesting application in bioinformatics is the determination of vector representations for protein sequences in order to infer physical or biological properties of unseen sequences through machine-learning models [9]. Differently from natural language processing, pre-trained representations for protein sequences are not commonly available and thus we need to train representations from the scratch.

Here, following the study in [9], a sequence of amino acids is divided into k lists of non-overlapping k-mers. For example, for $k = 3$, the amino acid sequence "ADTIVAVET" gives rise to 3 lists of 3-mers: "ADT, IVA, VET", "DTI,VAV" and "TIV, AVE". In this way, each amino acid sequence is viewed as a document whose phrases are the lists and words are the k-mers.

In [9], doc2vec [40] embedding models were trained on 524,529 protein sequences from UniProt database [41] and used to infer encondings of sequences for input to machine-learning algorithms for the end-task. Embeddings for the task sequences (not present in the training) were obtained by averaging the embeddings for the k lists of k-mers.

We consider two regression tasks named in [9] as "Localization" and "T50" (see [9] and references therein for details). For these tasks we used Gaussian process (GP) regression models [42] trained on the sequence embeddings given by doc2vec and compare its performance against the same GP model (using Matérn kernels with parameter $\nu = 5/2$) trained on embeddings obtained by the proposed DG-based methods.

We have used the same training and test sets[2] as in [9]. The context windows are defined by w k-mers before and after the central k-mer. We have used doc2vec from the Python library Gensim [43], with 25 epochs. In Table 4 we evaluate the predictions based on the coefficient of determination (R^2) and the mean absolute error (MAE). As

[1]Repository link

[2]The datasets and scripts are available at `https://github.com/fhalab/embeddings_reproduction/`

we can see, the results for embeddings obtained by GBU (with parameters $K = 64$, $m = 4K$) are slightly better than those from doc2vec with respect to these two measures. When it comes to training time, we remark that doc2vec spent more than 5 hours whereas GBU computed k-mers vector representations in less than 30 seconds.

5.2. Word embeddings in Natural Language Processing

The aim of this section is to contrast the training time of the proposed methods with those of popular ones for word representation while assessing their performance in intrinsic and extrinsic tasks. We shall see that the DG-based algorithms enjoy a remarkable reduction in training time and can reach competitive performances in extrinsic tasks (e.g. text classification).

To construct word representations, we used a corpus of 10^6 documents from Wikipedia 2016, which was cleaned using standard pre-processing methods in NLP (stop words and punctuation removal). The corpus is composed of $266,561,061$ tokens and the resulting vocabulary has $81,653$ distinct words. For the proposed DG-based methods, we used a window of $w = 10$ consecutive tokens for counting the co-occurrences.

The word vectors for PMI-eigs, GBU and DC were generated using Matlab [v.2018.b]. GloVe[3] and fastText[4] were compiled using GCC Apple LLVM version 10.0.0 (clang-1000.10.44.4) with the default parameters from the official code and trained in the corpus described above. We also compare these word representations with a baseline of random word vectors whose components are drawn from a standard Gaussian.

Table 5: Intrinsic evaluation (QVEC). First and second best are in bold and underline, respectively.

Dimension K	100	200	300
Random	14.89	21.82	27.08
GBU ($m = M = 4K$)	30.01	37.41	42.45
DC ($n_1 = 2 \times 10^4, n_{i\geq 2} = 800$)	34.21	41.42	45.69
PMI-eigs	**36.55**	**43.74**	**48.35**
Glove	35.43	42.06	46.25
fastText	<u>36.06</u>	<u>43.51</u>	<u>47.87</u>

Two experimental evaluations of word vectors were considered. First, an intrinsic evaluation, using QVEC [44], which was shown to have good correlation with the performance of the word vectors on semantic evaluation tasks, based on alignment of features extracted from lexical resources. These evaluations are reported in Table 5. Second, we evaluate the quality of these representations over three text classification tasks. Our implementation includes 3 datasets: WebKb (Multiclass), Subjectivity (Binary) and Amazon (Binary). Results are reported in Table 6.

[3]`https://github.com/stanfordnlp/GloVe`
[4]`https://github.com/facebookresearch/fastText/`

Table 6: F1 score - Text classification. First and second best are in bold and underline, respectively.

	$K = 100$			$K = 200$			$K = 300$		
Representation	Subject	WebKB	Amazon	Subject	WebKB	Amazon	Subject	WebKB	Amazon
Random	80.09	91.12	74.26	81.34	91.84	76.33	81.63	92.37	76.24
GBU ($m = M = 4K$)	**88.21**	93.04	80.58	88.05	**93.68**	81.39	**88.31**	93.40	82.36
DC ($n_1 = 20000, n_{i\geq 2} = 800$)	87.86	92.03	80.61	**88.28**	92.65	**82.49**	87.97	92.80	82.54
PMI-eigs	88.17	92.19	80.0	88.10	92.52	81.86	87.66	92.21	81.95
Glove	87.96	93.07	79.75	87.62	93.24	79.76	87.81	93.22	81.57
fastText	87.71	**93.37**	**80.64**	88.02	93.57	81.57	88.03	**93.82**	**82.64**

Table 7: Computing times (Left: Word Vectors, Right: Total). First and second best are in bold and underline, respectively.

	Dimensions		
Representations	100	200	300
Glove	1h07m	2h07m	2h42m
fastText	30m	48m	1h04m
PMI-Eigs	412s	836s	1088s
DC ($n_1 = 20000, n_{i\geq 2} = 800$)	348s	452s	618s
GBU ($m = M = 4K$)	**168s**	**188s**	**211s**

($K = 200$)	Datasets		
Represent. + Classif.	Subject	Amazon	WEBKB
Glove + CNN	7815s	8640s	9120s
fastText + CNN	3075s	3900s	4380s
PMI-Eigs + CNN	1031s	1856s	2336s
DC + CNN	602s	1368s	1952s
GBU + CNN	**383s**	**1208s**	**1688s**

For our classification experiments, we use an implementation of a Convolutional neural network (CNN) [45] using Tensorflow library [46] version 1.12. Overall, the performance in terms of F1-score between our word vectors, GloVe and fastText are very similar. A noticeable result is the performance of random embeddings on the WebKB dataset, which perform as well as the other representations. The reason for this behavior is that WebKB is made up of documents sharing lots of words in common when they belong to the same class. Therefore, the quality of the embeddings have less impact than the other tasks we consider.

We also provide some practical computing times, in the line of our complexity study. Table 7 reports the times for obtaining the word vectors (left) only, for dimensions $K = 100$, 200, 300, and the total time including the CNN training for $K = 200$ (right); parsing time is not included. Besides, for PMI-eigs, DC and GBU, we added the time for computing the matrix G (whose entries are given in (16)) from co-occurrence counts ($\approx 120s$) and for DC and GBU we also consider the time for computing the corresponding vertex order ($\approx 90s$ and $\approx 30s$, respec.). The parameters used in GBU were $M = m = 4K$ and the ones of DC were $n_1 = 20000$ and $n_{p\geq 2} = 800$.

From Table 7 we observe that the training times for the DG based methods are remarkably smaller than those of standard word vectors construction methods. They also improve the computational time with respect to the spectral decomposition of the whole PMI matrix. The price to be paid for these extremely fast word vectors, whose performance in text classification is close to well-established word embeddings such as GloVe and fastText (Table 6), is possibly an inferior performance in intrinsic tasks (Table 5).

6. Conclusion and perspectives

We proposed a formulation and methods based on Distance Geometry (DG) to generate vector representations of words from co-occurrence data. The resulting Geometric Build-up and Divide and Conquer algorithms are considerably faster than popular word embedding algorithms, as GloVe and fastText. Although the word vectors obtained by DG methods have performance inferior than the benchmarks in our intrinsic evaluation (QVEC), when combined with a convolutional neural network, DG word vectors lead to F1 scores among the best in three text classification tasks, but demanding about half to a quarter of the computing time, depending on the dataset. Concerning the regression tasks on protein sequences, the results obtained by DG-based methods were slightly better than those of doc2vec whereas the training time was extremely reduced. In our experiments, we considered a vertex order obtained by two heuristics: ranking the frequency of words in decreasing order for the GBU algorithm, and an order based on K-cores for the DC algorithm. The study of other vertex orders and their impact in the results of the GBU and DC algorithms are subject of future studies. Another question we would like to investigate is if the use of DG methods can also be extended for contextual embeddings.

Our approach based on distance geometry seems suitable for representation learning in low resources languages, or tasks in bioinformatics where pre-trained embeddings are not publicly available. Since the employed DG algorithms to obtain word vector representations scale well with the vocabulary size and dimension, they are adapted when working with limited computing resources. Therefore, Distance Geometry seems a promising paradigm for representation learning and perhaps other applications in data science.

References

[1] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: Y. Bengio, Y. LeCun (Eds.), 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013.

[2] J. Pennington, R. Socher, C. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), ACL, Doha, Qatar, 2014, pp. 1532–1543.

[3] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the Conference of the North American Chapter of the ACL, volume Human Language Technologies 1 of *NAACL*, ACL, Minneapolis, 2019, pp. 4171–4186.

[4] L. Liberti, A new distance geometry method for constructing word and sentence vectors, in: Companion Proceedings of the Web Conference 2020, WWW '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 679–685. doi:10.1145/3366424.3391267.

[5] F. Rousseau, M. Vazirgiannis, Graph-of-word and tw-idf: new approach to ad hoc ir, in: Proceedings of CIKM, New York, 2013.

[6] I. Dokmanic, R. Parhizkar, J. Ranieri, M. Vetterli, Euclidean distance matrices: Essential theory, algorithms, and applications, Signal Processing Magazine, IEEE 32 (2015) 12–30.

[7] L. Liberti, C. Lavor, N. Maculan, A. Mucherino, Euclidean distance geometry and applications, SIAM Review 56 (2014) 3–69.

[8] D. Jurafsky, J. H. Martin, Speech and Language Processing (2nd Edition), Prentice-Hall, Inc., USA, 2009.

[9] K. K. Yang, Z. Wu, C. N. Bedbrook, F. H. Arnold, Learned protein embeddings for machine learning, Bioinformatics 34 (2018) 2642–2648.

[10] P. Carpentier, G. Cohen, Vue d'ensemble de la méthode du gradient stochastique, in: Décomposition-coordination en optimisation déterministe et stochastique, Springer, 2017, pp. 167–193.

[11] L. Bottou, F. E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Review 60 (2018) 223–311.

[12] Q. Dong, Z. Wu, A linear-time algorithm for solving the molecular distance geometry problem with exact inter-atomic distances, Journal of Global Optimization 22 (2002) 365–375.

[13] J. B. Saxe, Embeddability of weighted graphs in k-space is strongly NP-hard, in: Proceedings of 17^{th} Allerton Conference in Communications, Control and Computing, Monticello, IL, 1979, pp. 480–489.

[14] R. Vidal, Y. Ma, S. Sastry, Generalized Principal Component Analysis, Springer, New York, 2016.

[15] S. Al-Homidan, H. Wolkowicz, Approximate and exact completion problems for euclidean distance matrices using semidefinite programming, Linear Algebra and its Applications 406 (2005) 109–141.

[16] I. J. Schoenberg, Remarks to Maurice Frechet's article: Sur la definition axiomatique d'une classe d'espaces vectoriels distancies applicables vectoriellement sur l'espace de Hilbert, Annals of Mathematics 36 (1935) 724–732.

[17] J. C. Gower, Euclidean distance geometry, Mathematical Scientist 7 (1982) 1–14.

[18] C. Eckart, G. Young, The approximation of one matrix by another of lower rank, Psychometrika 1 (1936) 211–218.

[19] N. Higham, Computing a nearest symmetric positive semidefinite matrix, Linear Algebra and its Applications 103 (1988) 103–118.

[20] A. Cassioli, O. Günlük, C. Lavor, L. Liberti, Discretization vertex orders in distance geometry, Discrete Applied Mathematics 197 (2015) 27–41.

[21] C. Lavor, J. Lee, A. Lee-St. John, L. Liberti, A. Mucherino, M. Sviridenko, Discretization orders for distance geometry problems, Optimization Letters 6 (2012) 783–796.

[22] D. Wu, Z. Wu, An updated geometric build-up algorithm for solving the molecular distance geometry problem with sparse distance data, Journal of Global Optimization 37 (2007) 661–673.

[23] G. H. Golub, C. F. Van Loan, Matrix Computations (3rd Ed.), Johns Hopkins University Press, USA, 1996.

[24] J. Aspnes, T. Eren, D. K. Goldenberg, A. S. Morse, W. Whiteley, Y. R. Yang, B. D. O. Anderson, P. N. Belhumeur, A theory of network localization, IEEE Transactions on Mobile Computing 5 (2006) 1663–1678. doi:`10.1109/TMC.2006.174`.

[25] V. Batagelj, M. Zaversnik, An $O(m)$ algorithm for cores decomposition of networks, arXiv preprint cs/0310049 (2003).

[26] P. H. Schönemann, A generalized solution of the orthogonal Procrustes problem, Psychometrika 31 (1966) 1–10.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[28] A. Globerson, G. Chechik, F. Pereira, N. Tishby, Euclidean embedding of co-occurrence data, Journal of Machine Learning Research 8 (2007) 2265–2295.

[29] P. D. Turney, P. Pantel, From frequency to meaning: Vector space models of semantics, Journal of Artificial Intelligence Research 37 (2010) 141–188.

[30] K. W. Church, P. Hanks, Word association norms, mutual information, and lexicography, Computational Linguistics 16 (1990) 22–29.

[31] O. Levy, Y. Goldberg, Neural word embedding as implicit matrix factorization, in: Advances in Neural Information Processing Systems, 2014, pp. 2177–2185.

[32] B. H. Soleimani, S. Matwin, Fast pmi-based word embedding with efficient use of unobserved patterns, Proceedings of the AAAI Conference on Artificial Intelligence 33 (2019) 7031–7038. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/4683`. doi:`10.1609/aaai.v33i01.33017031`.

[33] Z. Assylbekov, R. Takhanov, Context vectors are reflections of word vectors in half the dimensions, Journal of Artificial Intelligence Research 66 (2019) 225–242.

[34] S. Khalife, D. Gonçalves, Y. Allouah, L. Liberti, Further results on latent discourse models and word embeddings, Journal of Machine Learning Research 22 (2021) 1–36. URL: `http://jmlr.org/papers/v22/20-1413.html`.

[35] S. Arora, Y. Li, Y. Liang, T. Ma, A. Risteski, A latent variable model approach to PMI-based word embeddings, Transactions of the ACL 4 (2016) 385–399.

[36] T. B. Hashimoto, D. Alvarez-Melis, T. S. Jaakkola, Word embeddings as metric recovery in semantic spaces, Transactions of the ACL 4 (2016) 273–286.

[37] D. C. Sorensen, Implicit application of polynomial filters in a k-step arnoldi method, SIAM Journal on Matrix Analysis and Applications 13 (1992) 357–385.

[38] R. Lehoucq, D. Sorensen, C. Yang, ARPACK Users' Guide, SIAM, Philadelphia, PA, 1998.

[39] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Transactions of the ACL 5 (2017) 135–146.

[40] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: Proceedins of the 31st International Conference on Machine Learning, volume 32, 2014, pp. 1188–1196.

[41] T. U. Consortium, Uniprot: the universal protein knowledge-base, Nucleic Acids Res. 45 (2017) 158–169.

[42] C. Rasmussen, C. Williams, Gaussian Processes for Machine Learning, The MIT Press, Cambridge, MA, USA, 2006.

[43] R. Rehurek, P. Sojka, Software framework for topic modelling with large corpora, in: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, University of Malta, Valletta, Malta, 2010, pp. 45–50.

[44] Y. Tsvetkov, M. Faruqui, W. Ling, G. Lample, C. Dyer, Evaluation of word vector representations by subspace alignment, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015, pp. 2049–2054.

[45] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), ACL, Doha, Qatar, 2014, pp. 1746–1751.

[46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[47] S. J. Reddi, A. Hefny, S. Sra, B. Póczós, A. Smola, Stochastic variance reduction for nonconvex optimization, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, JMLR.org, 2016, p. 314–323.

[48] S. Ghadimi, G. Lan, Stochastic first- and zeroth-order methods for nonconvex stochastic programming, SIAM Journal on Optimization 23 (2013) 2341–2368.

[49] A. Khaled, P. Richtárik, Better theory for SGD in the nonconvex world, 2020. arXiv:2002.03329.

Appendix A. Random instances for GBU and SGD

The algorithm below describes how the synthetic data for experiments in Section 3.5 was generated.

Algorithm 3 Synthetic data generation for problem (12)

```
Input: integer $n$ (number of points), integer $K$ (dimension), float $\eta$ (noise relative amplitude)
Output (Pseudo-) Gram Matrix $G \in \mathbb{R}^{n\times n}$
1: $V = (V_1, \cdots, V_n) \leftarrow n$ samples from $\mathcal{N}(\mathbf{0}, \mathsf{Id}_{K\times K})$
2: Center $V$: $\forall i \in \{1, \cdots, n\}, V(:,i) \leftarrow V(:,i) - \frac{1}{n}\sum_{j=1}^{n} V(:,j)$
3: $G^{\mathsf{true}} \leftarrow V^T V$
4: $G \leftarrow \mathbf{0} \in \mathbb{R}^{n\times n}$
5: for $i = 1, \ldots, n$ do
6:     $\epsilon_{ii} \leftarrow$ sample from $\mathcal{N}(0, \eta \times G^{\mathsf{true}}_{ii})$
7:     $G_{ii} \leftarrow \max(0, G^{\mathsf{true}}_{ii} + \epsilon_{ii})$
8:     for $j = i+1, \ldots, n$ do
9:         $\epsilon_{ij} \leftarrow$ sample from $\mathcal{N}(0, \eta \times G^{\mathsf{true}}_{ij})$
10:        $G_{ij} \leftarrow G^{\mathsf{true}}_{ij} + \epsilon_{ij}$
11:        $G_{ji} \leftarrow G_{ij}$
12:    end for
13: end for
14: return $G$
```

Appendix B. Complexity of SGD for DGP

One may wonder about the complexity of SGD when applied to problem (12), the same problem to be solved by GBU. According to [47], for non-convex smooth objectives, SGD ensures a ε-stationary point V, i.e $\mathbb{E}[\|\nabla g(V)\|^2] \leq \varepsilon$, in $O(1/\varepsilon^2)$ iterations[5] [48]. Better iteration complexities are possible if the objective function g satisfies the Polyak-Łojasiewicz (PL)[6] or other growth conditions [49]. Unfortunately, although the objective $g(V)$ in (12) has Lipschitz gradient, it fails to verify the PL condition. Thus, considering the $O(1/\varepsilon^2)$ iteration complexity, if we work with a moderate tolerance, e.g $\varepsilon = 10^{-6}$ ($\|\nabla g(V)\| \leq 10^{-3}$), then in the worst case we need $O(10^{12})$ calls to the first order oracle (SGD iterations), which for dimension $K = 50$, incurs in computational cost of $\mathcal{C}_{SGD} \approx 5 \times 10^{13}$ (five orders of magnitude greater than $\mathcal{C}_{GBU}$ in the same setting). Even if variance reduction versions of SGD are considered, the computational complexity may still be higher than that of GBU. For instance, if SVRG [47] is used, with complexity $O(N + N^{2/3}/\varepsilon)$, for the objective in (12), $N = m(m+1)/2 + (n-m)m \approx O(mn)$, implying, for $m = 200$, $n = 10^5$,

[5] calls to the incremental first-order oracle;

[6] g satisfies the PL growth condition if there exists $\mu > 0$ such that $g(V) - g(V^\star) \leq (1/2\mu)\|\nabla g(V)\|^2$, where $V^\star$ is a global minimizer.

$K = 50$ and $\varepsilon = 10^{-6}$, that $N^{2/3}/\varepsilon = 5\sqrt[3]{20^2} \times 10^{11}$, which is already three orders of magnitude greater than $\mathcal{C}_{GBU}$. These numbers are consistent with the experiments reported in Section 3.5.