Efficient distributed representations beyond negative sampling

Lorenzo Dall'Amico[◇], Enrico Maria Belliardo

ISI Foundation, Via Chisola 5, 10126, Turin, Italy

October 30, 2023

Abstract

This article describes an efficient method to learn distributed representations, also known as *embeddings*. This is accomplished minimizing an objective function similar to the one introduced in the `Word2Vec` algorithm and later adopted in several works. The optimization computational bottleneck is the calculation of the *softmax* normalization constants for which a number of operations scaling quadratically with the sample size is required. This complexity is unsuited for large datasets and *negative sampling* is a popular workaround, allowing one to obtain distributed representations in linear time with respect to the sample size. Negative sampling consists, however, in a change of the loss function and hence solves a different optimization problem from the one originally proposed. Our contribution is to show that the sotfmax normalization constants can be estimated in linear time, allowing us to design an efficient optimization strategy to learn distributed representations. We test our approximation on two popular applications related to word and node embeddings. The results evidence competing performance in terms of accuracy with respect to negative sampling with a remarkably lower computational time.

Keywords— Representation learning, softmax approximation, negative sampling, Word2Vec, Node2Vec

1 Introduction

The ability of assessing the similarity of objects is one of the most crucial roles of machine learning [6]. This task is particularly hard to accomplish when dealing with complex entities – such as the words appearing in a text – that do not have an easily defined encoding. Distributed representations (or *embeddings*) allow one to map them into points of a high dimensional space, providing an answer to this problem [7]. In the aforementioned example, an effective representation maps *similar* words into nearby vectors.

A very relevant contribution to the field is the `Word2Vec` algorithm proposed by Mikolov *et al.* in [41] that owes its success to the efficiency, scalability and flexibility. Since its proposal, in fact, several works improved it [9] or drew inspiration from it to learn distributed representations of sentences and documents [41], graphs [49, 23, 20, 53, 47, 43], time [31], temporal contact sequences [22, 51, 46, 54, 59], biological entities [18, 45], tweets [17] and higher order interactions [8] among others. These algorithms build upon two steps: (1) for each $i \in \mathcal{V}$ – the set of entities – a sample $\mathcal{S}_i \subseteq \mathcal{V}$ is defined; (2) the set of embedding vectors $\{\boldsymbol{x}\}$ in $\mathbb{R}^d$ is obtained maximizing the log-likelihood $\mathcal{L}$,

[◇]Corresponding to: `lorenzo.dallamico@isi.it`

$$\mathcal{L}(\{\boldsymbol{x}\}) = \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{S}_i} \log\, p_{\boldsymbol{x}}(j|i)$$
$$p_{\boldsymbol{x}}(j|i) = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{x}_j}}{\sum_{k\in\mathcal{V}} e^{\boldsymbol{x}_i^T \boldsymbol{x}_k}} := \frac{e^{\boldsymbol{x}_i^T \boldsymbol{x}_j}}{Z_i}. \tag{1}$$

Intuitively, if j is often sampled from i, then the probability $p_{\boldsymbol{x}}(j|i)$ is large and the embedding vectors $\boldsymbol{x}_i, \boldsymbol{x}_j$ become closer, obtaining a similar representation to related entities. The known limitation of the optimization of this cost function resides in the partition function Z_i, requiring $\mathcal{O}(n)$ operations to be computed, where $n = |\mathcal{V}|$ is the number of items to be embedded. Since Z_i has to be computed for all $i \in \mathcal{V}$, optimizing $\mathcal{L}$ requires $\mathcal{O}(n^2)$ operations, making it unsuited to approach large datasets. One of the most commonly adopted workarounds is *negative sampling* [41] consisting in replacing $\mathcal{L}$ with the cost function $\tilde{\mathcal{L}}$ of a weighted logistic principal component analysis [34]. Its minima should be close to those of $\mathcal{L}$ but it has the advantage of being computed in a linear time with respect to n. Negative sampling has been extensively adopted but, to date, it is still only poorly understood from a theoretical standpoint and some studies evidenced its potential weaknesses [12, 50, 34, 42].

In this paper we propose an approximation method that allows one to estimate all the Z_i's in $\mathcal{O}(n)$ operations. Based on this result, we define the embedding problem in a very general way, introducing a cost function similar to the one appearing in Equation (1) with the addition of a regularization term. This way our proposed algorithm can be easily deployed in a variety of contexts. In particular, we suppose there exists a set of sparse sampling probability vectors $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$ in $\mathbb{R}^n$ encoding a measure of proximity between i and the other elements of $\mathcal{V}$. For instance, in the case of words, $p_j^{(i)}$ may indicate the probability that j is in the same sentence as i. The design of the set $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$ is problem-dependent and beyond the scope of this article in which we describe an efficient framework to create a distributed entity representation *given* the probability set. Our main contributions are:

1. We provide a theoretical framework showing that the set of *all* partition functions $\{Z_i\}_{i\in\mathcal{V}}$ can be accurately estimated in $\mathcal{O}(n)$ operations. Based on this result, we describe an efficient strategy to obtain distributed representations.
2. We show that the pure `Python` implementation of our algorithm is competitive or outperforms in terms of speed an optimized `C` implementation of `Word2Vec`.[1]
3. We display two use cases of our algorithm applied to node and words embeddings, proving its efficacy also in terms of accuracy.
4. We provide `Python` implementation of our algorithm at github.com/lorenzodallamico/EDRep.

Notations

- Sets are denoted with the font $\mathcal{A}$. The cardinality of a set is denoted with $|\mathcal{A}|$.
- Matrices are denoted with a standard capital letter, A. The identity matrix of size n is I_n.
- The notation $a_n = O_n(b_n)$ implies that there exists a finite c so that $\lim_{n\to\infty} \frac{a_n}{b_n} = c$. Similarly $a_n = o_n(b_n)$ implies that $\lim_{n\to\infty} \frac{a_n}{b_n} = 0$.
- Vectors are indicated in boldface (e.g. $\boldsymbol{a}$), while vector elements and scalars in standard font (e.g. a_i). The all-ones vector of size n is denoted with $\mathbf{1}_n$.

2 Main result

2.1 Problem position

Consider a set of items $\mathcal{V}$ with $n = |\mathcal{V}|$. For all $i \in \mathcal{V}$, we let $\boldsymbol{p}^{(i)} \in \mathbb{R}^n$ be a probability distribution defined on $\mathcal{V}$, encoding similarity: a large $p_j^{(i)}$ implies high affinity between i and j. We further define $\boldsymbol{p}_0 \in \mathbb{R}^n$

[1] We refer to the *Gensim* implementation [52], available at radimrehurek.com/gensim/ run in parallel on 8 cores.

as a baseline $\boldsymbol{p}^{(i)}$ should be compared to. We aim at giving a *distributed representation* of $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$, *i.e.* a vector embedding $\{\boldsymbol{x}_i\}_{i\in\mathcal{V}}$ that preserves structural properties encoded in the probability distributions. We equivalently denote the set $\{\boldsymbol{x}_i\}_{i\in\mathcal{V}}$ as $X \in \mathbb{R}^{n\times d}$, a matrix storing the embedding vectors in its rows. We define X as the solution to the following optimization problem:

$$X = \arg\min_{Y\in\mathcal{U}_{n\times d}} \sum_{i\in\mathcal{V}} \Bigg[\underbrace{-\mathbb{E}_{\boldsymbol{p}^{(i)}}\left(\log \frac{\boldsymbol{q}_Y^{(i)}}{Z_i}\right)}_{\text{cross-entropy}} + \underbrace{\mathbb{E}_{\boldsymbol{p}_0}\left(\log \boldsymbol{q}_Y^{(i)}\right)}_{\text{regularization}} \Bigg]$$
$$q_{Y,j}^{(i)} = e^{\boldsymbol{y}_i^T \boldsymbol{y}_j}; \qquad Z_i := \sum_{k\in\mathcal{V}} e^{\boldsymbol{y}_i^T \boldsymbol{y}_k}, \tag{2}$$

where $\mathcal{U}_{n\times d}$ is the set of all matrices of size $n\times d$ having in their rows unitary vectors. In words, we adopt a variational approach, approximating $\boldsymbol{p}^{(i)}$ with the distribution $\boldsymbol{q}_Y^{(i)}$ and optimize over the embeddings. The cross-entropy term is the same appearing in Equation (1) and causing the $\mathcal{O}(n^2)$ complexity. To this we only add the regularization "promotes" the probability part exceeding the baseline, thus preventing the embedding to encode a trivial piece of information.

Remark 1 (Extension to an asymmetric setting)**.** *In Equation (2) and in the remainder, we only consider distributions $\boldsymbol{p}^{(i)}$ that are defined and indexed over the same set, $\mathcal{V}$. It is however possible to consider an asymmetric scenario in which the entries $p_a^{(i)}$ are defined for $i \in \mathcal{V}$ and $a \in \mathcal{W}$. In this case, one needs to define two embedding matrices $X \in \mathbb{R}^{|\mathcal{V}|\times d}$ and $Y \in \mathbb{R}^{|\mathcal{W}|\times d}$ and let*

$$q_{X,Y,a}^{(i)} = e^{\boldsymbol{x}_i^T \boldsymbol{y}_a}; \qquad Z_i = \sum_{a\in\mathcal{W}} e^{\boldsymbol{x}_i^T \boldsymbol{y}_a}.$$

All the framework described below has a straightforward generalization to this setting that we discuss with an example of application in Appendix A.2.

Given Equation (2) our goal is to efficiently compute Z_i for all $i \in \mathcal{V}$ to obtain the embedding X that solves the optimization problem. We now proceed describing our main result on an efficient way to estimate the normalization constants.

2.2 Efficient estimation of Z_i

Our main result consists in a closed formula heuristic approximation for the estimation of the Z_i's appearing in Equation (2). To obtain this result, we treat the embedding vectors as random variables and perform three steps: first we show that Z_i is concentrated towards its expected value; then, under proper hypotheses that the expected value converges to an integral form; finally we provide a heuristic argument to approximate and solve analytically the integral, obtaining an explicit expression for the Z_i's. The proofs of the first two steps, enunciated in Proposition 1, 2 are given in Appendix A.1.

Proposition 1. *Consider a set $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ of n independent unitary random vectors in $\mathbb{R}^d$. Letting Z_i as per Equation (2), then for all $t > 0$*

$$\mathbb{P}\left(\left|\frac{Z_i}{n} - \mathbb{E}\left[\frac{Z_i}{n}\right]\right| \geq t\right) \leq 4e^{-(\sqrt{n}t/4e)^2}.$$

The values Z_i/n are random variables but Proposition 1 states that for all large n they asymptotically converge to their expected value that, instead, is deterministic, thus allowing us to simplify the calculations. We now state Proposition 2 that provides an explicit formula for the expectation of Z_i/n in the large n limit.

Proposition 2. *Under the same assumptions of Proposition 1, consider an arbitrary i fixed and assume that for all $j \neq i$, $\boldsymbol{x}_i^T \boldsymbol{x}_j$ converges in distribution to a variable $\omega_i \sim f_i$, then,*

$$\lim_{n\to\infty} \mathbb{E}\left[\frac{Z_i}{n}\right] = \int_{\frac{1}{e}}^{e} dt\; e^t f_i(t).$$

The non-trivial statement of Proposition 2 is due to the fact that the convergence in distribution does not imply the convergence of the expectations, but it does under these hypotheses. In general, however, no expression for the distribution f_i can be given and thus the integral cannot be solved. We now introduce a heuristic argument to conveniently approximate the Z_i values. Specifically, assuming that both d and n are large, we replace the function f_i with a mixture of κ Gaussians:

$$\int_{\frac{1}{e}}^{e} dt\ e^t f_i(t) \approx \int_{\mathbb{R}} dt\ e^t \sum_{a=1}^{\kappa} \pi_a\ \mathcal{N}(t; \boldsymbol{x}_i^T \boldsymbol{\mu}_a, \boldsymbol{x}_i^T \Omega_a \boldsymbol{x}_i) \tag{3}$$

where $\boldsymbol{\mu}_a \in R^d$, $\Omega_a \in R^{d \times d}$ and $\mathbf{1}_\kappa^T \boldsymbol{\pi} = 1$. Equation 3 has three fundamental aspects that need to be commented. Firstly, we do not assume a "point-wise" convergence of f_i to a mixture of Gaussians, but only that the two integrals are close. Secondly, in Equation (3) we move from an integral on a finite set $[e^{-1}, e]$ to one over all real axis. We claim this can be done safely because the integrand function goes to zero at least as fast as $e^{-|t|}$ and consequently the contribution from the tails of the integral is expected to be negligible. The key point, however, lies in the choice of the mixture of Gaussians approximation. This follows from the scalar product being a sum of random variables. If the embedding dimension d is large enough, we expect its distribution to be well approximated by a Gaussian. Moreover, using $\kappa > 1$ allows one to assume that not all j follow the same distribution and that there may exist clusters in the embedding. The mean and variances expressions follow from the assumptions of Proposition 2 in which i is fixed and j is a random variable. Note that this approximation does not require that the embedding vectors are, themselves, drawn from a multivariate Gaussian distribution, since it only focuses on the scalar product distribution. Given the approximation of Equation (3) and combining the results of Proposition 1, 2, we are now in position to formulate our main result.

Z_i estimation formula. Consider the normalization constants Z_i of the optimization problem formulated in Equation (2). Then, for $\kappa \geq 1$ and a set of parameters $\{\pi_a, \boldsymbol{\mu}_a, \Omega_a\}_{a=1,\dots,\kappa}$ one can write

$$\frac{Z_i}{n} \approx \sum_{a=1}^{\kappa} \pi_a\ e^{\boldsymbol{x}_i^T \boldsymbol{\mu}_a + \frac{1}{2} \boldsymbol{x}_i^T \Omega_a \boldsymbol{x}_i}. \tag{4}$$

The remarkable advantage of Equation (4) is that, given the parameters $\{\pi_a, \boldsymbol{\mu}_a, \Omega_a\}_{a=1,\dots,\kappa}$, Z_i is computed in $\mathcal{O}(\kappa d^2)$ operations, independently of n, thus allowing us to estimate all Z_i's in $\mathcal{O}(n)$ operations provided that the parameters can also be obtained in linear time with respect to n. While potentially sub-optimal, we observed that a simple clustering-based approach practically performs well to achieve this task, as shown in the empirical validation of Section 2.3. First of all, consider the trivial case $\kappa = 1$. Then, $\boldsymbol{\mu}, \Omega$ would trivially be the mean vector and covariance matrix of the embedding and $\pi = 1$. For $\kappa > 1$ we split the dataset into clusters defining a labeling function $\ell : \mathcal{V} \to \{1, \dots, \kappa\}$ and define $\boldsymbol{\mu}_a, \Omega_a$ as the mean vector and covariance matrix of the embedding vectors of class a, *i.e.* $\{\boldsymbol{x}_i\ :\ \ell(i) = a\}$. The value π_a is related to the size of class a, $\pi_a = \frac{1}{n}|\{i\ :\ \ell(i) = a\}|$. The label assignment is obtained with *k-means* [39, 13], allowing us to obtain the parameter estimation in $O(n\kappa d^2)$ operations that indeed keeps the complexity linear in n.

Let us now proceed with an empirical validation of our main result.

2.3 Empirical validation

We consider 4 datasets taken from the NLPL word embeddings repository [33], representing word embeddings obtained with different algorithms:

0. *British National Corpus*, trained with *Continuous Skip-Gram* [41], $n = 163.473$, $d = 300$;
7. *English Wikipedia Dump 02/2017*, trained with *Global Vectors* [48], $n = 273.930$, $d = 300$;
9. *English Wikipedia Dump of 02/2017*, trained with *fastText Skipgram* [29], $n = 273.930$, $d = 300$;
224. *Ukrainian CoNLL17 corpus*, trained with *Continuous Bag-of-Words* [41], $n = 99.884$, $d = 200$.

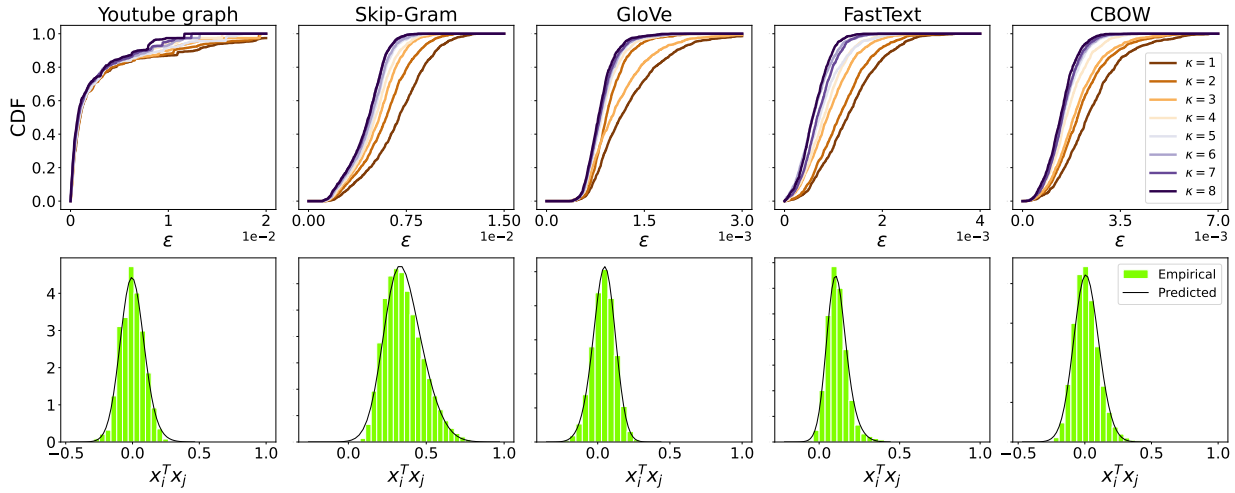

Figure 1: **Empirical validation of Equation** (4). *Top row*: cumulative density function (CDF) of the relative error ϵ between the true an estimated value of Z_i for 1000 randomly selected indices i and the order of the mixture of Gaussian distribution $\kappa = 1, \ldots, 8$, color-encoded. *Bottom row*: histogram of $\boldsymbol{x}_i^T \boldsymbol{x}_j$ for a fixed i in green and in black the mixture of Gaussian fit, for $\kappa = 5$.

We further include a fifth dataset, constituted by the embedding obtained applying on a real-world graph our proposed `EDRep` algorithm that solves the optimization problem of Equation (2) (see Section 3.1 for further details). For each of these datasets we (1) set the norm of each embedding vector to 1, (2) sample at random 1000 words and (3) compute the corresponding exact and the estimated Z_i values for different approximation orders κ. Figure 1 shows the results of this procedure: in the top row the cumulative density function (CDF) of the relative error is displayed for several values of κ. As expected, the precision is an increasing function of κ, but very good approximations are obtained on all datasets for $\kappa = 1$ already. Note that in this case, the clustering step can be omitted and is, therefore, of particular interest. The bottom row shows the empirical and the predicted distribution of the $\boldsymbol{x}_i^T \boldsymbol{x}_j$ for a randomly chosen fixed i and $\kappa = 5$, confirming the goodness of the approximation.

2.4 `EDRep`: efficient distributed representations

Let us now detail the main steps needed to translate the result of Equation (4) into a practical algorithm to produce efficient distributed representations.

Optimization strategy

According to Equation (2), the embedding matrix $X \in \mathbb{R}^{n \times d}$ is obtained by optimizing a cost function $\mathcal{L}(X)$ that, after some trivial calculations, can be rewritten in the following form:

$$\mathcal{L}(X) = -\sum_{i \in \mathcal{V}} \left[\sum_{j \in \mathcal{V}} \left(p_j^{(i)} - p_{0,j} \right) \boldsymbol{x}_i^T \boldsymbol{x}_j + \log(Z_i) \right]. \tag{5}$$

We now exploit the result of Equation (4) and make the notation more compact. Let $P \in \mathbb{R}^{n \times n}$ be a matrix so that $P_{ij} = p_j^{(i)}$. Then, $\mathcal{L}(X)$ can be rewritten as:

Algorithm 1: `EDRep` optimization

Input: $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$, set of probabilities; $\boldsymbol{p}_0 \in \mathbb{R}^n$ null model probability; d, embedding dimension; $\boldsymbol{\ell} \in \{1,\dots,\kappa\}^n$ node label vector; η_0, learning rate; `n_epochs`, number of training epochs

Output: $X^{n\times d}$, embedding matrix

begin
- $X \leftarrow$ initialize the embedding matrix;
- $\eta \leftarrow \eta_0$ initial learning rate;
- **for** $1 \le t \le$ *n_epochs* **do**
  - $\boldsymbol{\mu}_{a=1,\dots,\kappa}$, $\Omega_{a=1,\dots,\kappa} \leftarrow$ update the parameters;
  - $\{\boldsymbol{g}_i\}_{i\in\mathcal{V}} \leftarrow$ gradient matrix as in Equation (6);
  - **for** $1 \le i \le n$ **do**
    - $\boldsymbol{g}_i' \leftarrow \boldsymbol{g}_i - (\boldsymbol{g}_i^T\boldsymbol{x}_i)\boldsymbol{x}_i$ remove the parallel component;
    - $\boldsymbol{g}_i'' \leftarrow \boldsymbol{g}_i'/\|\boldsymbol{g}_i'\|$ normalize;
    - $\boldsymbol{x}_i \leftarrow \sqrt{1-\eta^2}\,\boldsymbol{x}_i - \eta\boldsymbol{g}_i''$; gradient descent step;
  - **end**
  - $\eta \leftarrow \eta - \frac{\eta_0}{\text{n\_epochs}}$ linear update of the learning rate;
- **end**

end

$$\begin{aligned}\mathcal{L}(X) &= -\mathrm{tr}\left[X^T\left(P - \mathbf{1}_n\boldsymbol{p}_0^T\right)X\right] + \sum_{i\in\mathcal{V}}\log\sum_{a=1}^{\kappa}\underbrace{\pi_a\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\mu}_a + \frac{1}{2}\boldsymbol{x}_i^T\Omega_a\boldsymbol{x}_i\right\}}_{Z_{ia}}\\ &:= -\mathrm{tr}\left[X^T\left(P - \mathbf{1}_n\boldsymbol{p}_0^T\right)X\right] + \sum_{i\in\mathcal{V}}\log\left(Z\mathbf{1}_\kappa\right)_i\end{aligned}$$

where we also introduced the matrix $Z \in \mathbb{R}^{n\times\kappa}$ with elements Z_{ia} for convenience. The proposed strategy to optimize $\mathcal{L}$ is summarized in Algorithm 1 and is a constrained gradient descent. Letting $M \in \mathbb{R}^{d\times\kappa}$ have the $\{\boldsymbol{\mu}_a\}_{a=1,\dots,\kappa}$ values in its rows, the $i \in \mathcal{V}$ and $\alpha \in \{1,\dots,d\}$ gradient component reads

$$g_{i,\alpha} = -[(P+P^T)X]_{i\alpha} + [\mathbf{1}_n(\boldsymbol{p}_0^TX) + \boldsymbol{p}_0(\mathbf{1}_n^TX)]_{i\alpha} + \frac{1}{Z_i}\left([ZM]_{i\alpha} + \sum_{a=1}^{\kappa}Z_{ia}[X\Omega_a]_{i\alpha}\right). \tag{6}$$

To keep the normalization, we first remove the component parallel to $\boldsymbol{x}_i$, then we normalize the vector and obtain $\boldsymbol{g}_i''$ to finally update the embedding as follows for some learning rate $0 \le \eta \le 1$:

$$\boldsymbol{x}_i^{\text{new}} = \sqrt{1-\eta^2}\,\boldsymbol{x}_i - \eta\boldsymbol{g}_i''$$

that implies $\|\boldsymbol{x}_i^{\text{new}}\| = 1$. Note that in Algorithm 1, the labeling vector $\boldsymbol{\ell}$ is required as an input but it is generally not known in advance. A workaround we implemented consists in running first the algorithm for $\kappa = 1$ for which $\boldsymbol{\ell} = \mathbf{1}_n$, the run κ-class *k-means* and rerun `EDRep` algorithm for the so-obtained vector $\boldsymbol{\ell}$.

Computational complexity of the algorithm

The cost of the optimization procedure described in Algorithm 1 requires $\mathcal{O}(n\kappa d)$ operations to compute $\boldsymbol{\ell}$, $\mathcal{O}(n\kappa d^2)$ for the "Z part" of the gradient, and $\mathcal{O}(Ed)$ to compute the product PX, where E denotes the

number of non-zero entries of P. This complexity becomes prohibitive when $E \gg n$, but this is not the case in typical settings and the product can be performed very efficiently. Nonetheless, even when E is very large, if P can be written as the product (or sum of products) of sparse matrices, PX can still be computed very efficiently. In fact, let $P = P_m \cdot P_{m-1} \dots P_1$ for some positive m, then PX can be obtained without computing P, simply taking the products from right to left:

$$PX = (P_m \cdot \overleftarrow{P_{m-1} \cdot \dots} \cdot P_1 X),$$

thus speeding up the computational bottleneck of our algorithm. In our implementation we explicitly consider this representation of P as an input. When a non-factorized dense matrix P is provided, one could envision to adopt a method such as the one presented in [36] to approximate a dense matrix P with the product of sparse matrices to considerably speed up the algorithm.

Let us now proceed showing two use cases of our algorithm, applied to the generation of node and word distributed representations. In Appendix A.2 we further discuss the extension of Algorithm 1 to the asymmetric setting mentioned in Remark 1 with a further use case of our algorithm for matrix completion.

3 Two use cases

In this section we analyze two use cases for two popular applications of distributed representations: node and words embeddings. In the former case, the goal is to determine a distributed representation of the vertices of a graph that preserves its structural properties [11, 21]. On the other hand, distributed word representations are one of the most popular applications of this class of algorithms that allow one to encode words semantic similarity [2]. In both cases, pairs of *similar* items (nodes or words) should correspond to embedding vectors with a high cosine similarity.

Remark 2. *To test our algorithm on practical applications, we must define the set* $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$. *In the remainder, we adopt simple strategies to obtain this set and show, nonetheless, that the* `EDRep` *approach achieves competitive results in terms of performance. We would like to underline, however, that the sampling probabilities choice is a hard and problem-dependent task and addressing it optimally is beyond the scope of this article. Moreover, we formulated* `EDRep` *as an* offline *algorithm while* `Word2Vec` *is* online. *This difference influences the algorithm performances but not the essence our main result – Equation 4 – that provides an efficient estimation of the partition functions* regardless *of the details of the implementation strategy.*

3.1 Node embeddings

A graph is a tuple $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of *nodes* and $\mathcal{E}$ the set of edges connecting nodes [44, 10]. Graphs are a versatile tool to model interacting systems and have a range spectrum of application, including social networks [55, 62], technological networks [3], spatial networks [5], biological networks [1] and many others. A graph $\mathcal{G}$ can be easily represented in matrix form with the *adjacency matrix* $A \in \mathbb{R}^{n\times n}$, so that $A_{ij} = 1$ if $(ij) \in \mathcal{E}$ and equals zero otherwise. Here and in the remainder we only focus on undirected ($A_{ij} = A_{ji}$) and unweighted graphs ($A_{ij} \in [0, 1]$). Our algorithm can, however, deal with the challenging directed and weighted cases, provided that an appropriate sampling strategy is defined.

One of the most relevant problems in graph learning is *community detection* that can be informally defined as the task of determining a non-overlapping node partition, unveiling more densely connected groups of nodes [19]. In the following – similarly to what done in *spectral clustering* [60] – we adopt the `EDRep` algorithm to produce a node embedding and then perform clustering in the embedded space. Our choice of P in this case is

$$P = \frac{1}{w} \sum_{t=1}^{w} (L_{\text{rw}})^t , \tag{7}$$

where L_{rw} is the *random-walk Laplacian*, obtained from A and imposing that its rows sum up to 1. The entry P_{ij} of this matrix is the limiting probability that a random walker on $\mathcal{G}$ goes from node i to j in one w or fewer steps. We further set $\boldsymbol{p}_0 = \mathbf{1}_n/n$, that is equivalent to comparing $\boldsymbol{p}^{(i)}$ with a uniform prior.

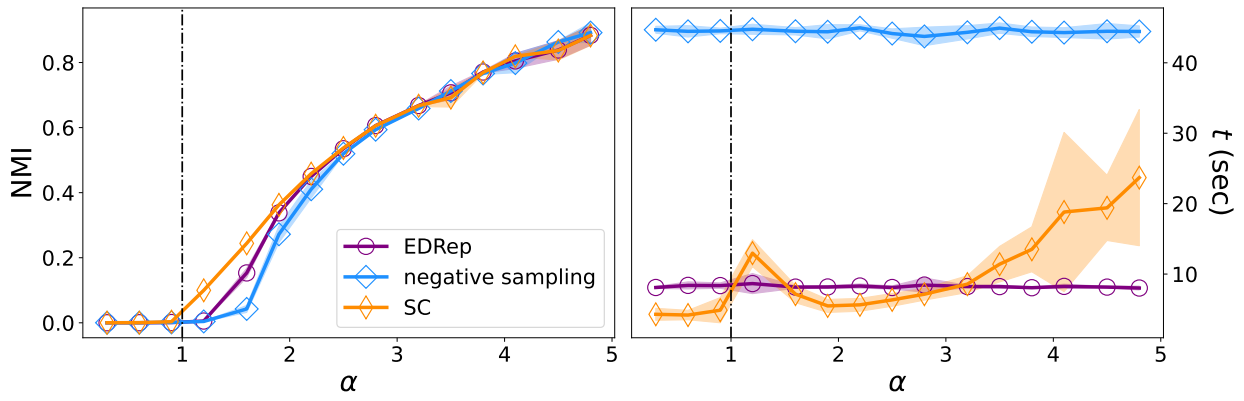

Figure 2: **Community detection on synthetic graphs**. *Left panel*: NMI score as a function of α (3) for a DCSBM graph (Definition 1) with $n = 30\,000$, $c = 10$, $q = 4, \kappa = 1$ and varying values of c_{out}. The three lines correspond to three different embedding techniques. The purple dots are the `EDRep` algorithm with $d = 32$, the cyan diamonds is `DeepWalk` with $d = 32$ for P as in Equation (7) with $w = 3$, while the orange narrow diamonds are the spectral clustering algorithm of [15]. *Right panel*: corresponding computational time in logarithmic scale. *Both plots*: averages are over 10 samples and the error bar width equals the standard deviation.

With these choices, we generate a distributed node representation X and then cluster the nodes of $\mathcal{G}$ into groups. To evaluate the accuracy we consider synthetic graphs generated from the *degree-corrected stochastic block model* in which a ground truth label assignment is known and real-world graphs with a class annotation. The accuracy is measured in terms of *normalized mutual information* (NMI), a score that ranges between 0 (random label assignment) and 1 (perfect assignment).

Synthetic graphs

The *degree-corrected stochastic block model* (DCSBM) [30] is a random graph generative model capable to create graphs with a community structure and a arbitrary degree distribution.[2] This model is convenient to study community detection as it provides a clear definition of communities but it also allows one to model the broad degree distribution, typically observed in real-world graphs [4].

Definition 1 (DCSBM)**.** *Let $\omega : \mathcal{V} \to \{1, \dots, q\}$ be a class labeling function, where q is the number of classes. Let $\mathbb{P}(\omega_i = a) = q^{-1}$ and consider two positive integers satisfying $c_{\text{in}} > c_{\text{out}} \geq 0$. Further Let $\theta \sim p_\theta$ be a random variable that encodes the intrinsic node connectivity, so that $\mathbb{E}[\theta] = 1$ and $\mathbb{E}[\theta^2] = O_n(1)$. For all $i \in \mathcal{V}, \theta_i$ is drawn independently at random from p_θ. The entries of the graph adjacency matrix are generated independently (up to symmetry) at random with probability*

$$\mathbb{P}(A_{ij} = 1) = \frac{\theta_i \theta_j}{n} \cdot \begin{cases} c_{\text{in}} & \text{if } \omega(i) = \omega(j) \\ c_{\text{out}} & \text{else} \end{cases}$$

In words, nodes in the same community ($\omega(i) = \omega(j)$) are connected with a higher probability than nodes in different communities. From a straightforward calculation, the expected degree is $\mathbb{E}[c_i] \propto \theta_i$. For this model, the goal is to infer the node label assignment from a realization of A. It was shown that this is theoretically feasible (in the large n regime) if and only if a function of the model parameters[3] $\alpha > 1$ [25, 24]. In left panel of Figure 2 we show the NMI score obtained by different algorithms as a function of α, while the right panel shows the computation time in logarithmic scale. We compare:

- the spectral algorithm of [16, 14, 15] that was shown to be nearly Bayes-optimal for this task,

[2] The degree of a node is its number of connections.

[3] Letting c be the expected average degree $c = \frac{c_{\text{in}} + (k-1)c_{\text{out}}}{k}$, $\alpha = (c - c_{\text{out}})\sqrt{\frac{\mathbb{E}[\theta^2]}{c}}$.

Name	n	q	EDRep – NMI	DW – NMI	EDRep – t	DW – t
amazon	9 317	232	0.94	0.93	3 ± 0.4 s	16.6 ± 0.8 s
youtube	26 931	608	0.61	0.65	9.3 ± 0.7 s	64.3 ± 2.7 s
dblp	60 530	252	0.55	0.50	20.8 ± 2.3 s	145 ± 6.6 s
livejournal	64 504	1662	0.91	0.91	30.2 ± 2.1 s	166.9 ± 12.6 s

Table 1: **Community detection on real graphs**. For each graph we report: 1) the name; 2) the size n; 3) the number of communities q; 4,5) the average NMI score for `EDRep` and `DeepWalk` [49] (DW) and 6,7) the corresponding average computational time in seconds with the respective deviation. The results are obtained over 10 simulations. The NMI scores do not specify a standard deviation because, in all cases, it was below 0.01.

- an algorithm combining the `EDRep` optimization approach with P as in Equation (7) and $\kappa = 1$ and *k-means*,
- an algorithm combining `DeepWalk` [49] with *k-means*. We used the `C++` implementation available at github.com/thibaudmartinez/node2vec. The algorithm is deployed with its default values.

The results show that the `EDRep`-based algorithm performs almost as well as the optimal algorithm of [15] and a slight mismatch is only observed for hard problems, *i.e.* for α approaching 1. This is a particularly challenging setting in which few algorithms are capable to retrieve the community structure. The method of [15] was explicitly designed to work well in this regime in which, instead, the spectral algorithm based on the *random walk Laplacian* [57] (that has the same eigenvectors of our choice of P) performs poorly. When compared to the `DeepWalk` approach, our method generally yields better results for all α. On top of accuracy, the computational time gives the `EDRep` approach a decisive advantage with respect to the `C++` efficient implementation of `DeepWalk` being more than 4 times faster with a improvement observed also with respect to the spectral clustering algorithm. The main advantage in this case is however related to the algorithm's computational complexity. The considered spectral clustering algorithm runs in $\mathcal{O}(nq^3)$ operations, while the complexity of `EDRep` is independent of q.[4] Consequently, for large q, spectral clustering has to pay a significant price in terms of computational efficiency and it becomes unpractical.

Real-world graphs

We further test our algorithm on more realistic data available from the Stanford Network Analysis Platform [38] that come with a community label annotation. In most cases, for the considered datasets, each node may be associated to more than one community. For simplicity, in order to keep the same framework as the one detailed above, each node was associated to the largest community it belongs to. In Table 1 we report the results summarizing the network properties, the obtained NMI and the computational time. In this case, spectral clustering is not considered since its complexity does not allow us to obtain a partition in a reasonable time. Similar to what already mentioned for DCSBM-generated graphs, the `EDRep` algorithm obtains competing performances with respect to negative sampling at a much lower computational time.

3.2 Word embeddings

Word embeddings are one of the most prominent applications for the class of algorithms `EDRep` belongs to. The goal is to encode in a vector-representation the words semantic similarity as inferred from a text [56]. For instance, the words *flower* and *petal* are closely related and should be represented by vectors with a high cosine similarity. This representation is mathematically convenient and opens up to a great span of applications such as those discussed in [61, 32] to provide some examples.

Unlike graphs, for word representations the matrix P is not given and it has to be built from the text. To do so we define a co-occurrence matrix $W \in \mathbb{R}^{n \times n}$, where n denotes the vocabulary size and W_{ij} is the

[4] A relevant setting is the one in which $\kappa = q$, *i.e.* the order of the GMM approximation equals the number of communities. It this case, `EDRep`'s complexity scales as $\mathcal{O}(q^2)$. It has to be said, however, that for very large q it seems pointless to set $\kappa = q$ since $\kappa \approx 8$ gave very good estimates in all cases we tested. Actually, while the estimation of Z_i is more accurate for large κ, we did not see any significant gain in the overall performance when choosing $\kappa > 1$.

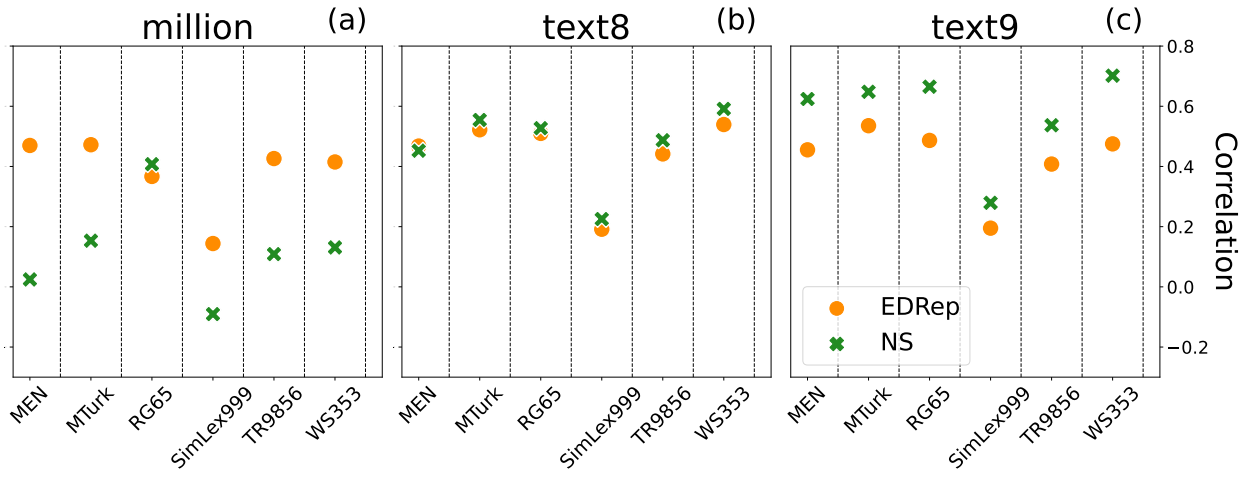

Figure 3: **Test on word embeddings**. Performance achieved on different benchmarks indicated on the x axis for the `million`, `text8`, `text9` datasets. The results of the `EDRep` algorithm (orange dot) are compared with `Word2Vec` (green cross). The plotted quantity is average Spearman correlation between the benchmark similarity scores and those obtained with the two embedding methods. Averages are over 20 samples for `million` and `text8` and over 10 samples for `text9`. Reading the plots from left to right (a, b, c) the *online* negative sampling approach profits from longer training unlike the *offline* `EDRep` method, as detailed in Section 3.2 and motivated in Remark 2.

number of times the word j appeared in the *context* of word i, *i.e.* within a maximum distance from i. For example, in Albus Dumbledore's sentence

It takes a great deal of bravery to stand up to our enemies,
but just as much to stand up to our friends

the words *stand, up, to, our, but, just, as, much* appear in a context of size 4 around *enemies*. Given W and normalizing its rows we obtain our choice for the matrix P in which large entries correspond to pairs of words frequently appearing close to one-another. As one can observe in the example above, however, some context-independent words tend to appear with high frequency without carrying much information. This is the case of "*to*" that has four occurrences in the short sentence. Prior to building the co-occurrence matrix, we first proceed with the standard cleaning procedure introduced in [41] in which each word is dropped with a probability $1 - \sqrt{e^{-5}/f_i}$, where f_i denotes the number of appearances of word i in the text. For the $\boldsymbol{p}_0$ vector, we drew inspiration from negative sampling and used $p_{0,i} \propto f_i^\gamma$ with $\gamma = 0.75$. A standard way to evaluate word embeddings is to check if the cosine similarity between pairs of words relates to its semantic affinity. To obtain a quantitative performance measure, we used the datasets made available by [28] in which a list of pairs of words (a, b) are associated to a score s_{ab}. For each of these pairs, we compute the cosine similarity obtained from the embeddings and use the Spearman correlation coefficient to quantify the agreement between the two measures. Higher correlation values indicate a good match with manually annotated data. The results are compared in Figure 3.2 with the `Word2Vec` algorithm using $d = 32$ as embedding dimension. In particular, Figure 3.2b evidences a neat advantage of the `EDRep` approach when dealing with a small text, as it allows us to learn reasonable word representations from a very short text already. Increasing the input file size, in `text8` (Figure 3.2c) we observe very similar performances for the two algorithms, while on `text9` (Figure 3.2d), higher performances are obtained by `Word2Vec` on all benchmarks and `EDRep` performs essentially as well as on `text8`.

These results can be easily interpreted when looking more closely to the two algorithms. `EDRep` must first learn a matrix text representation and then a distributed representation thereof. Consequently, the algorithm training time is independent of the text length. On the opposite, `Word2Vec`, learns the embeddings in an *online* fashion and longer texts imply more training epochs. These two algorithms are indeed different and an online training method may effectively be more powerful than an offline one for this specific task. With

this perspective, given that our main result of Equation 4 would still hold, one could envision to adapt `EDRep` to different training strategies. Moreover, `EDRep`'s ability to obtain significant representation from sparse data could be exploited to pre-train the `Word2Vec` embeddings on a short text to speed up convergence.

Coming to the computation time, we confirm also in this case that our approach is faster on all three texts by approximately a factor 2, even though `Word2Vec` is run in parallel on all available cores. One can notice that the gain in computational efficiency is smaller than the one obtained in Section 3.1. This is because in that case a graph readily comes with a matrix representation and `Node2Vec` has to generate a "text" from the graph. Contrariwise, here the matrix must be obtained from the text and, with our implementation, it requires approximately 50% of the total time.

As a final remark, we would like to stress that the scores reported in Figure 3.2 are an only marginally objective way to evaluate a word embedding. In fact, we tried other design choices of the matrix P and in many cases we ended up obtaining larger scores on some benchmarks but lower on others. For this reason, we comfortably state that, once a proper evaluation metric is defined, there is room for improvement in the design of the P matrix and these results should be interpreted as an algorithm deployment example.

4 Conclusion

In this article we showed that, under proper conditions, an approximation of the normalization constants of *softmax* can be obtained in linear time. This allowed us to optimize a loss function similar to the one originally proposed in [41] to efficiently obtain distributed representations. We detailed a theoretical framework justifying our claim that was validated on real data and then translated our result into a practical algorithm. This method relies on the definition of a probability matrix P capable to encode the similarity between the entities to be represented. To foster the impact of our work, we showed two simple examples applied to node and word embeddings.

In both cases, our proposed approach proved to be faster than an optimized negative sampling `C` implementation. This is a striking result, but we are firmly convinced that there is a lot of room for improvement. In fact, our implementation is fully coded in `Python` – that is known to be slower than `C` –, it is not parallelized and it can reasonably gain a considerable advantage if carefully optimized. Moreover, our main result fits in a larger frame than the one described in this paper and can potentially be exploited with other optimization strategies, as we already suggested along the lines.

Negative sampling appeared to be the natural benchmark of comparison of our work, but we would like to stress that it consists in optimizing a different cost function from the one optimized in `EDRep`. We noticed a remarkable difference between the case in which an affinity matrix representation is given (graphs) from the one in which it is not (texts). In the earlier case all the information related to the corpus (the graph) is represented in the form of a sparse matrix, hence the *offline* implementation of `EDRep` works optimally and significantly outperformed `Node2Vec` both in terms of speed and of accuracy. In the latter case, when encoding the text into a matrix, all the complex relations between the words should be captured by the matrix structure. This is a non-trivial task that might penalize infrequent co-occurrences between pairs of words. We believe, however, that a crucial advantage of our approach lies in its high interpretability. For a given practical deployment of our algorithm, a practitioner only needs to define a sampling strategy that meaningfully encodes proximity for the problem at hand. The simplicity of our framework makes it also easier to study than negative sampling with the techniques described in [27], for instance. Moreover, our main result holds in a wider range of settings with respect to the one explored and it can be easily generalized to non symmetric, or non normalized P matrices.

Acknowledgments LD acknowledges support from the the Lagrange Project of the ISI Foundation funded by CRT Foundation and from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101016233 (PERISCOPE) and from Fondation Botnar. EMB acknowledges support from CRT Lagrange Fellowships in Data Science for Social Impact of the ISI Foundation. The authors thank Ciro Cattuto for fruitful discussions during the project conception and Cosme Louart for his valuable feedback and help on the proof of Propositions 1, 2.

References

[1] Eric Alm and Adam P Arkin. Biological networks. *Current opinion in structural biology*, 13(2):193–202, 2003.

[2] Felipe Almeida and Geraldo Xexéo. Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*, 2019.

[3] Luis A Nunes Amaral, Antonio Scala, Marc Barthelemy, and H Eugene Stanley. Classes of small-world networks. *Proceedings of the national academy of sciences*, 97(21):11149–11152, 2000.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[5] Marc Barthélemy. Spatial networks. *Physics reports*, 499(1-3):1–101, 2011.

[6] Aurélien Bellet, Amaury Habrard, and Marc Sebban. Metric learning. *Synthesis lectures on artificial intelligence and machine learning*, 9(1):1–151, 2015.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] Jacob Charles Wright Billings, Mirko Hu, Giulia Lerda, Alexey N Medvedev, Francesco Mottes, Adrian Onicas, Andrea Santoro, and Giovanni Petri. Simplex2vec embeddings for community detection in simplicial complexes. *arXiv preprint arXiv:1906.09068*, 2019.

[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[10] Stephen P Borgatti and Daniel S Halgin. On network theory. *Organization science*, 22(5):1168–1181, 2011.

[11] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering*, 30(9):1616–1637, 2018.

[12] Long Chen, Fajie Yuan, Joemon M Jose, and Weinan Zhang. Improving negative sampling for word representation using self-embedded features. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 99–107, 2018.

[13] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[14] Lorenzo Dall'Amico, Romain Couillet, and Nicolas Tremblay. Revisiting the bethe-hessian: improved community detection in sparse heterogeneous graphs. In *Advances in Neural Information Processing Systems*, pages 4039–4049, 2019.

[15] Lorenzo Dall'Amico, Romain Couillet, and Nicolas Tremblay. A unified framework for spectral clustering in sparse graphs. *Journal of Machine Learning Research*, 22(217):1–56, 2021.

[16] Lorenzo Dall'Amico and Romain Couillet. Community detection in sparse realistic graphs: Improving the bethe hessian. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2942–2946. IEEE, 2019.

[17] Bhuwan Dhingra, Zhong Zhou, Dylan Fitzpatrick, Michael Muehl, and William W. Cohen. Tweet2vec: Character-based distributed representations for social media. *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Short Papers*, page 269 – 274, 2016. doi: 10.18653/v1/p16-2044.

[18] Jingcheng Du, Peilin Jia, Yulin Dai, Cui Tao, Zhongming Zhao, and Degui Zhi. Gene2vec: distributed representation of genes based on co-expression. *BMC genomics*, 20(1):7–15, 2019.

[19] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics reports*, 659:1–44, 2016.

[20] Zheng Gao, Gang Fu, Chunping Ouyang, Satoshi Tsutsui, Xiaozhong Liu, Jeremy Yang, Christopher Gessner, Brian Foote, David Wild, Ying Ding, et al. edge2vec: Representation learning using edge semantics for biomedical knowledge discovery. *BMC bioinformatics*, 20(1):1–15, 2019.

[21] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[22] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, 2020.

[23] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[24] Lennart Gulikers, Marc Lelarge, and Laurent Massoulié. A spectral method for community detection in moderately sparse degree-corrected stochastic block models. *Advances in Applied Probability*, 49(3):686–721, 2017.

[25] Lennart Gulikers, Marc Lelarge, and Laurent Massoulié. An impossibility result for reconstruction in the degree-corrected stochastic block model. *The Annals of Applied Probability*, 2018.

[26] Folasade Olubusola Isinkaye, Yetunde O Folajimi, and Bolande Adefowoke Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian informatics journal*, 16(3):261–273, 2015.

[27] Ariel Jaffe, Yuval Kluger, Ofir Lindenbaum, Jonathan Patsenker, Erez Peterfreund, and Stefan Steinerberger. The spectral underpinning of word2vec. *Frontiers in applied mathematics and statistics*, 6:593406, 2020.

[28] Stanisław Jastrzebski, Damian Leśniak, and Wojciech Marian Czarnecki. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *arXiv preprint arXiv:1702.02170*, 2017.

[29] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April 2017.

[30] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[31] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

[32] Faiza Khan Khattak, Serena Jeblee, Chloé Pou-Prom, Mohamed Abdalla, Christopher Meaney, and Frank Rudzicz. A survey of word embeddings for clinical text. *Journal of Biomedical Informatics*, 100:100057, 2019.

[33] Andrei Kutuzov, Murhaf Fares, Stephan Oepen, and Erik Velldal. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 58th Conference on Simulation and Modelling*, pages 271–276. Linköping University Electronic Press, 2017.

[34] Andrew J Landgraf and Jeremy Bellay. Word2vec skip-gram with negative sampling is a weighted logistic pca. *arXiv preprint arXiv:1705.09755*, 2017.

[35] Monique Laurent. Matrix completion problems. *Encyclopedia of Optimization*, 3:221–229, 2009.

[36] Luc Le Magoarou and Rémi Gribonval. Flexible multilayer sparse approximations of matrices and applications. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):688–700, 2016.

[37] Michel Ledoux. *The concentration of measure phenomenon*. American Mathematical Soc., 2001.

[38] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[39] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[40] WAJ Luxemburg. Arzela's dominated convergence theorem for the riemann integral. *The American Mathematical Monthly*, 78(9):970–979, 1971.

[41] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[42] David Mimno and Laure Thompson. The strange geometry of skip-gram with negative sampling. In *Empirical Methods in Natural Language Processing*, 2017.

[43] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[44] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[45] Patrick Ng. dna2vec: Consistent vector representations of variable-length k-mers. *arXiv preprint arXiv:1701.06279*, 2017.

[46] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion proceedings of the the web conference 2018*, pages 969–976, 2018.

[47] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.

[48] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[49] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[50] Pengda Qin, Weiran Xu, and Jun Guo. A novel negative sampling based on tfidf for learning word representation. *Neurocomputing*, 177:257–265, 2016.

[51] Mahmudur Rahman, Tanay Kumar Saha, Mohammad Al Hasan, Kevin S Xu, and Chandan K Reddy. Dylink2vec: Effective feature representation for link prediction in dynamic networks. *arXiv preprint arXiv:1804.05755*, 2018.

[52] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[53] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. Gemsec: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining*, pages 65–72, 2019.

[54] Koya Sato, Mizuki Oka, Alain Barrat, and Ciro Cattuto. Predicting partially observed processes on temporal networks by dynamics-aware node embeddings (dyane). *EPJ Data Science*, 10(1), 2021. doi: 10.1140/epjds/s13688-021-00277-8.

[55] John Scott. Trend report social network analysis. *Sociology*, pages 109–127, 1988.

[56] S Selva Birunda and R Kanniga Devi. A review on word embedding techniques for text classification. *Innovative Data Communication Technologies and Application: Proceedings of ICIDCA 2020*, pages 267–281, 2021.

[57] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[58] Michel Talagrand. Concentration of measure and isoperimetric inequalities in product spaces. *Publications Mathématiques de l'Institut des Hautes Etudes Scientifiques*, 81:73–205, 1995.

[59] Maddalena Torricelli, Márton Karsai, and Laetitia Gauvin. weg2vec: Event embedding for temporal networks. *Scientific Reports*, 10(1):1–11, 2020.

[60] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

[61] Yanshan Wang, Sijia Liu, Naveed Afzal, Majid Rastegar-Mojarad, Liwei Wang, Feichen Shen, Paul Kingsbury, and Hongfang Liu. A comparison of word embeddings for the biomedical natural language processing. *Journal of biomedical informatics*, 87:12–20, 2018.

[62] Stanley Wasserman and Katherine Faust. Social network analysis: Methods and applications. *Cambridge university press*, 1994.

[63] Roman Zykov, Noskov Artem, and Anokhin Alexander. Retailrocket recommender system dataset, 2022. URL https://www.kaggle.com/dsv/4471234.

Appendix

A.1 Proofs

In this appendix we proofs of Proposition 1 and 2, which we also report here for convenience.

Proposition 1. *Consider a set $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ of n independent unitary random vectors in $\mathbb{R}^d$. Letting Z_i as per Equation (2), then for all $t > 0$*

$$\mathbb{P}\left(\left|\frac{Z_i}{n} - \mathbb{E}\left[\frac{Z_i}{n}\right]\right| \geq t\right) \leq 4e^{-(\sqrt{n}t/4e)^2}.$$

Before proceeding with the proof of Proposition 1, let us first enunciate the following concentration theorem that we will use in our demonstration.

Theorem 1 ([58], [37], Corollary 4.10)**.** *Given a random vector $\boldsymbol{\omega} \in [-1, 1]^n$ with independent entries and a 1-Lipschitz (for the euclidean norm) and convex mapping $g : \mathbb{R}^n \to \mathbb{R}$, one has the concentration inequality:*

$$\forall\ t > 0 : \quad \mathbb{P}\left(|g(\boldsymbol{\omega}) - \mathbb{E}[g(\boldsymbol{\omega})]| \geq t\right) \leq 4e^{-t^2/16}.$$

Proof (Proposition 1)**.** *Let $\boldsymbol{\omega}^{(i)} \in \mathbb{R}^{n-1}$ be a vector with entries $\{\boldsymbol{x}_i^T \boldsymbol{x}_j\}_{j \in \mathcal{V}\setminus\{i\}}$. Then, the vector $\boldsymbol{\omega}^{(i)}$ satisfies the hypothesis of Theorem 1 for all i. We now define the function g as follows:*

$$g(\boldsymbol{w}^{(i)}) = \frac{1}{n} \sum_{j \in \mathcal{V}\setminus i} e^{w_j^{(i)}},$$

then one can immediately verify that $Z_i = e + ng(\boldsymbol{\omega}^{(i)})$. We are left to prove that g satisfies the hypotheses of Theorem 1 as well. Firstly, g is convex since it is the sum of convex functions. We now compute the Lipschitz parameter. Considering two vectors $\boldsymbol{\omega}, \boldsymbol{\omega}' \in [-1, 1]^n$ one can give the following bound for any $i \in \mathcal{V}$:

$$\begin{aligned}\left|g(\boldsymbol{\omega}) - g(\boldsymbol{\omega}')\right| &\overset{(a)}{\leq} \frac{1}{n} \sum_{j \in \mathcal{V}\setminus i} \left|e^{\omega_j} - e^{\omega'_j}\right| \overset{(b)}{\leq} \frac{e}{n} \sum_{j \in \mathcal{V}\setminus i} \left|\omega_j - \omega'_j\right| \overset{(c)}{=} \frac{e}{n} \cdot \mathbf{1}_{n-1}^T |\boldsymbol{\omega} - \boldsymbol{\omega}'| \\ &\overset{(d)}{\leq} \frac{e}{n} \cdot \|\mathbf{1}_{n-1}\| \cdot \|\boldsymbol{\omega} - \boldsymbol{\omega}'\| = \frac{e\sqrt{n-1}}{n} \|\boldsymbol{\omega} - \boldsymbol{\omega}'\| \leq \frac{e}{\sqrt{n}} \|\boldsymbol{\omega} - \boldsymbol{\omega}'\|,\end{aligned}$$

where in (a) we used the triangle inequality, in (b) we exploited the fundamental theorem of calculus, in (c) the $|\cdot|$ is meant entry-wise and finally in (d) we used the Cauchy-Schwartz inequality. This set of inequalities implies that $\frac{\sqrt{n}g}{e}$ is 1-Lipschitz and is a suitable choice to apply Theorem 1. Proposition 1.1 is then easily obtained from a small play on t and exploiting the relation between Z_i and $g(\boldsymbol{\omega}^{(i)})$.

We now recall here the enunciation of Proposition 2.

Proposition 2. *Under the same assumptions of Proposition 1, consider an arbitrary i fixed and assume that for all $j \neq i$, $\boldsymbol{x}_i^T \boldsymbol{x}_j$ converges in distribution to a variable $\omega_i \sim f_i$, then,*

$$\lim_{n\to\infty} \mathbb{E}\left[\frac{Z_i}{n}\right] = \int_{\frac{1}{e}}^{e} dt\ e^t f_i(t).$$

Also in this case, before proceeding with the proof, let us enunciate the dominated convergence theorem that allows one to invert the integral and limit signs.

Theorem 2 ([40])**.** *Let $F_1, \dots, F_n$ be a sequence of Riemann-integrable functions – defined on a bounded and closed interval $[a, b]$ – which converges on $[a, b]$ to a Riemann-integrable function F. If there exists a constant $m > 0$ satisfying $|F_n(t) \leq M|$ for all $x \in [a, b]$ and for all n, then*

$$\lim_{n\to\infty} \int_a^b dt F_n(t) = \int_a^b dt \lim_{n\to\infty} F_n(t) = \int_a^b dt\ F(t).$$

Proof (Proposition 2)**.** *The values $\boldsymbol{x}_i^T \boldsymbol{x}_j := \omega_j^{(i)}$ are a sequence of $n-1$ random variables with cumulative densities $F_{i,1}, \dots, F_{i,n}$.*

$$\begin{aligned}
\lim_{n\to\infty} \mathbb{E}\left[\frac{Z_i}{n}\right] &= \lim_{n\to\infty} \frac{1}{n}\left[e + \sum_{j\in\mathcal{V}\setminus i} \mathbb{E}\left[e^{\omega_j^{(i)}}\right]\right] = \lim_{n\to\infty} \frac{1}{n}\left[e + (n-1)\mathbb{E}\left[e^{\omega_n^{(i)}}\right]\right] \\
&= \lim_{n\to\infty} \mathbb{E}\left[e^{\omega_n^{(i)}}\right] \overset{(a)}{=} \lim_{n\to\infty} \int_{\frac{1}{e}}^{e} dt\ e^t f_{i,n} \overset{(b)}{=} \lim_{n\to\infty} \left[e^t F_{i,n}\Big|_{\frac{1}{e}}^{e} - \int_{\frac{1}{e}}^{e} dt\ e^t F_{i,n}(t)\right] \\
&\xrightarrow[n\to\infty]{(c)} e^t F_i(t)\Big|_{\frac{1}{e}}^{e} - \int_{\frac{1}{e}}^{e} dt\ e^t F_i(t) = \int_{\frac{1}{e}}^{e} dt\ e^t f_i(t).
\end{aligned}$$

In (a) we denoted with f the derivative of F, i.e. *the probability density function; in (b) we performed an integration by parts; in (c) we exploited the fact that convergence in distribution implies the pointwise convergence of the probability density function and we applied Theorem 2 choosing $M = e$.*

A.2 The asymmetric setting

As we mentioned in the main text, Algorithm 1 can be easily extended to a scenario in which the rows and the columns of the probability matrix P play a different role and, potentially, do not have the same size. As a reference, which will further explore as a use case, the matrix $P \in \mathbb{R}^{n\times m}$ could index users and products in its rows and columns, respectively. One can then obtain two embedding matrices $X \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^{m\times d}$ that are informative about the affinity of different users, based on their tastes; the affinity of different products, based on the users' interactions; the likelihood that a user will interact with a given item. This last task is at the basis of recommendation systems [26] and matrix completion [35]. Letting $\mathcal{V}, \mathcal{W}$ be the indices of the rows and columns respectively, we rewrite the loss function of Equation (5) as follows:

$$\begin{aligned}
\mathcal{L}(X, Y) &= -\sum_{i\in\mathcal{V}} \left[\sum_{j\in\mathcal{W}} \left(p_j^{(i)} - p_{0,j}\right) \boldsymbol{x}_i^T \boldsymbol{y}_j + \log(Z_i)\right], \\
Z_i &= \sum_{j\in\mathcal{W}} e^{\boldsymbol{x}_i^T \boldsymbol{y}_j}.
\end{aligned}$$

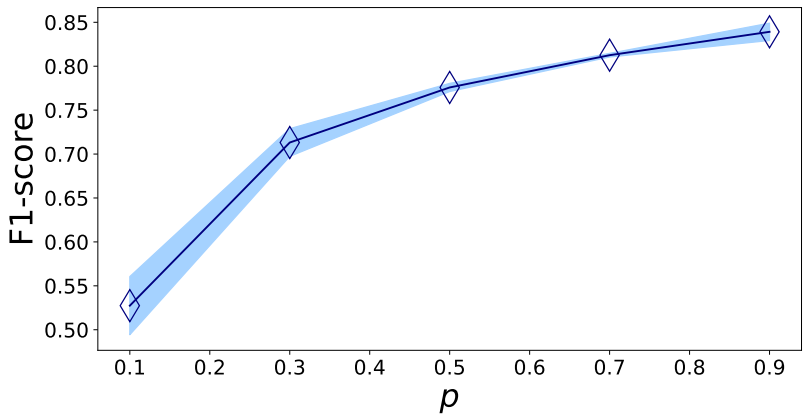

Figure A.1: **Test on recommendation system**: F-score performance as a function of the training set relative size, obtained following the procedure detailed in Section A.3. The results are averaged over 10 realizations and the (barely visible) shaded line indicates the variance over all experiments.

Now, one can easily verify that the results of Proposition 1 have a straightforward generalization to this setting, as well as Equation (4). The only difference is that the *k-means* step has to be run solely on the embedding matrix Y and that two gradient matrices have to be used to update the weights. With a self-explanatory notation, we obtain the gradient vectors $\{\boldsymbol{g}_i^{(X)}\}_{i\in\mathcal{V}}$ and $\{\boldsymbol{g}_a^{(Y)}\}_{a\in\mathcal{W}}$:

$$
\begin{aligned}
g_{i\alpha}^{(X)} &= -[(P - \mathbf{1}_n \boldsymbol{p}_0^T)Y]_{i\alpha} + \frac{1}{Z_i}\left([ZM]_{i\alpha} + \sum_{a=1}^{\kappa} Z_{ia}[X\Omega_a]_{i\alpha}\right) \\
g_{a\alpha}^{(Y)} &= -[(P^T - \boldsymbol{p}_0 \mathbf{1}_n^T)X]_{a\alpha}.
\end{aligned}
$$

Note that now $\boldsymbol{p}_0 \in \mathbb{R}^m$ and that to compute $\boldsymbol{g}_a^{(Y)}$ we neglected the gradient of the "Z-component" of the loss function that is proportional to $1/n$. Let us now provide an example of usage of the `EDRep` algorithm in this setting.

A.3 Use case: recommendation system

We worked with the freely available `RetailRocket recommender system` dataset [63]. This is composed by a sequence of events collected from a commerce website. Each event is associated with a time-stamp, a user id (all users form the set $\mathcal{V}$, according to our notation), an item id (all items for the set $\mathcal{W}$) and an event-type that can be "view", "add to cart", "transaction" that we naïvely map to the scalars $[1, 2, 3]$. To test our algorithm, we perform the following steps:

1. We split the dataset into a training and a test set, denoting with p the probability of belonging to the training set.
2. We then build a matrix $W \in \mathbb{R}^{n\times m}$ so that W_{ia} is the sum of all scores $(1, 2$ or $3)$ corresponding to user i and item a as they appear in the training set. The matrix $P \in \mathbb{R}^{n\times m}$ is then obtained from W normalizing its rows to one and represents a proxy of probability of interaction between i and a.
3. We use the matrix P as an input of `EDRep` to obtain two embedding matrices, X, Y.
4. We create a dataset E, in which half of the entries are taken from the test set and the other half are randomly drawn (ia) pairs.

5. For each (ia) pair in E we then attempt to recognize "good" from "fake" pairs, according to whether the dot product $\boldsymbol{x}_i^T \boldsymbol{y}_a$ is positive or negative.

Figure A.1 shows the $F1$ score obtained following the pipeline we just described for different values of p and evidences that the algorithm is capable of performing non-trivial recommendations for small training sets and that profits from larger ones, as expected.