Efficient distributed representations with linear-time attention scores normalization

Anonymous authors
Paper under double-blind review

Abstract

The attention score matrix $\mathrm{SoftMax}(XY^T)$ encodes relational similarity patterns between objects and is extremely popular in machine learning. However, the complexity required to calculate it runs quadratically with the problem size, making it a computationally heavy solution. In this article, we propose a linear-time approximation of the attention score normalization constants for embedding vectors with bounded norms. We show on several pre-trained embeddings that the accuracy of our estimation formula surpasses competing kernel methods by even orders of magnitude. From this result, we design a linear-time and task-agnostic embedding algorithm based on the optimization of the attention scores. The proposed algorithm is highly interpretable and easily adapted to an arbitrary embedding problem. We consider a few use-cases and observe similar or higher performances and a lower computational time with respect to comparable embedding algorithms.

1 Introduction

A foundational role of machine learning is to provide expressive representations of complex objects with vectors that preserve some properties of the represented data. These vectors, also known as *distributed representations* or *embeddings*, enable otherwise ill-defined operations on the data, such as assessing their similarity Bengio et al. (2013); Bellet et al. (2015). Relational patterns among the represented objects are a relevant example of a property that embedding vectors can encode. For instance, in language processing, distributed representations capture the semantic similarity between words, leveraging the patterns in which they appear in a text. This is possible, for instance, by training the attention score matrix $\mathrm{SoftMax}(XY^T)$, where X, Y are two embedding matrices, possibly with $X = Y$. This term ranges between 0 and 1 for each pair of indices, promoting similar representations for related objects. The optimization of the attention scores motivates several embedding algorithms such as `Word2Vec` Mikolov et al. (2013) and the many "`2Vec`"-type algorithms that followed. This term also appears in attention-based transformers, with great importance for several machine learning tasks Vaswani et al. (2017).

A known limitation of optimizing the attention scores lies in its computational complexity. For a system of n objects, computing this matrix requires $\mathcal{O}(n^2)$ operations, which is impractical for large datasets. Methods to reduce this complexity have a long history and are still actively explored, as reviewed in Tay et al. (2022). Since McFadden (1977), sampling was recognized as an effective workaround Bengio & Senécal (2003). One of the most popular methods is *negative sampling* Mikolov et al. (2013), considering *de facto* an alternative loss function that can be computed in $\mathcal{O}(n)$ operations. While it has been extensively studied and improved Mu et al. (2019); Rawat et al. (2019); Shan et al. (2018); Bamler & Mandt (2020), several works highlighted the weaknesses of sampling Landgraf & Bellay (2017); Chen et al. (2018); Qin et al. (2016); Mimno & Thompson (2017). Other methods rely on approximate, but easy-to-normalize forms of the attention score matrix Wang et al. (2020); Xiong et al. (2021) exploiting, for instance, the kernel trick Choromanski et al. (2020); Peng et al. (2021).

In this work, we provide a closed formula to compute in linear time the attention score normalization constants. We provide theoretical and empirical results supporting our method. Testing our approximation on several pre-trained embeddings, we observe a systematic gain in the accuracy of our method, even by

orders of magnitude, compared to approaches based on the kernel trick. We then build on this result to design an efficient embedding algorithm in the `2Vec` spirit. Given their good scalability and simplicity of design, these algorithms proved extremely useful for machine learning users and have been applied to various contexts. However, while their initial formulation describes a loss function containing the attention score matrix, negative sampling is often adopted and the proposed loss function is not actually optimized. We address this gap by showing how to efficiently optimize a cost function taking $\log[\mathrm{SoftMax}(XY^T)]$ as argument. This term can be decomposed into two parts: the numerator XY^T that is low rank and can consequently be efficiently optimized; the denominator involving the *softmax* normalization constants that is the computational bottleneck. Our estimation formula allows us to compute it in $\mathcal{O}(n)$ operations, and thus to efficiently optimize the embedding cost function. We then showcase a few practical applications evidencing that our algorithm is competitive or outperforms comparable methods in terms of speed and performance.[1] We recall that all these benchmark algorithms have a linear complexity in n, and do not optimize the softmax function as we do. We stress that we do not claim that sampling approaches are bad *per se*, but rather, we investigate how to efficiently optimize a cost function containing the softmax attention scores. A `Python` implementation of our algorithm is shared alongside the paper, in the supplementary material.

2 Main result

This section describes how to efficiently compute the normalization constants of the attention score matrix $\mathrm{SoftMax}(XY^T)$. Here $X \in \mathbb{R}^{n\times d}$, $Y \in \mathbb{R}^{m\times d}$ are two embedding matrices, with a potentially different number of rows, but with the same number of columns. In the remainder, we let $m = \mathcal{O}(n)$. We denote with $\{\boldsymbol{x}_i\}_{i=1,\dots,n}$ and $\{\boldsymbol{y}_a\}_{a=1,\dots,m}$ the sets of vectors contained in the rows of X and Y, respectively. The i-th attention score normalization constant reads

$$Z_i = \sum_{a=1}^{m} e^{\boldsymbol{x}_i^T \boldsymbol{y}_a} \ . \tag{1}$$

Computing all the Z_i's requires $\mathcal{O}(dnm) = \mathcal{O}(dn^2)$ operations, which scales quadratically with the problem size. In the following, we require the embedding vectors to have bounded norms and treat them as independent random vectors drawn from an unknown distribution. Note that this assumption does not imply the representation dimensions to be themselves independent thus preventing the possibility of representing complex similarity patterns. On the opposite, such patterns are captured by the underlying unknown probability distribution on which we make no assumption, besides requiring embedding vectors to have bounded norms. In Theorem 2.1 we provide a concentration result of Z_i/m around its expectation. This allows us to treat the normalization constants as deterministic variables. We then obtain an integral form for the expectation of Z_i, depending on the unknown distribution f_i of the scalar product $\boldsymbol{x}_i^T \boldsymbol{y}_a$. Finally, we introduce a multivariate Gaussian approximation of order κ to approximate the distribution f_i. We propose a heuristic strategy to efficiently obtain the Gaussian approximation and derive an explicit formula to estimate all the Z_i's of Equation (2). The empirical tests show that this approximation gives good results for small values of κ, making it useful in practical settings. The proofs of Theorem 2.1, 2.2 are given in Appendix A.

2.1 Linear-time attention score normalization

The following theorem provides a concentration result of the rescaled Z_i value around its mean. This simplifies the analysis, allowing us to substitute the attention score normalization with its expectation.

Theorem 2.1. *Consider a vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and a set $\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_m\}$ of m independent random vectors in $\mathbb{R}^d$. Let $|\boldsymbol{x}_i^T \boldsymbol{y}_a| \leq h = \mathcal{O}_m(1)$ for all a. Letting Z_i be defined as in Equation (1), then for all $t > 0$*

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

This result holds unchanged in the case $X = Y$. The proof can be easily adapted by singling out the negligible contribution given by $e^{\|\boldsymbol{x}_i\|^2} \leq e^h = \mathcal{O}_n(1)$, obtaining the same result. Theorem 2.1 states that,

[1] All codes are run on a Dell Inspiron laptop with 16 Gb of RAM and with a processor 11th Gen Intel Core i7-11390H @ 3.40GHz × 8.

for all large m, the random variables Z_i/m asymptotically converge to its expectation and can thus be treated as deterministic. We underline that, not only Theorem 2.1 does not make assumptions on the embedding vectors' distribution, but it also does not require them to be identically distributed. These general assumptions allow us to exploit Theorem 2.1 in practical tasks, as detailed in the remainder.

We enunciate Theorem 2.2, which provides a formula for the expectation of Z_i/m in the large m limit.

Theorem 2.2. *Under the same assumptions of Theorem 2.1, assume that for all a, $\boldsymbol{x}_i^T \boldsymbol{y}_a$ converges in distribution to a variable $\omega_i \sim f_i$, then,*

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

Theorem 2.2 assumes the loosest type of convergence and shows that under the assumptions of Theorem 2.1 it implies the convergence in expectation. The theorem can be easily extended for $X = Y$, also in this case. Combining Theorems 2.1, 2.2, in the large m limit, we thus obtain an integral expression to which Z_i/m converges, under mild assumptions. With no additional hypotheses, however, the integral can still not be solved. We thus proceed by introducing a parametric approximation of f_i, denoted with $\tilde{f}_i$. Thanks to Theorem 2.2, the goodness of this approximation only needs to hold in distribution sense, i.e. the cumulative density functions of f_i and $\tilde{f}_i$ should be close. We choose to write $\tilde{f}_i$ as a mixture of κ Gaussians since they are capable of approximating density functions with an arbitrary small error, provided κ is large enough Goodfellow et al. (2016). Recalling that f_i is the distribution of a scalar product – i.e. a sum of random variables – we expect that for d large enough, this distribution will be well approximated by the Gaussian mixture for even small κ values. As shown in Section 2.2, the empirical evidence confirms the goodness of this approximation on real data.

$$\int_{-h}^{h} dt\ e^t f_i(t) \approx \int_{\mathbb{R}} dt\ e^t \sum_{\alpha=1}^{\kappa} \pi_\alpha\ \mathcal{N}(t; \boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha, \boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i)\ ,$$

where $\boldsymbol{\mu}_a \in \mathbb{R}^d$, $\Omega_a \in \mathbb{R}^{d\times d}$ and $\mathbf{1}_\kappa^T \boldsymbol{\pi} = 1$. The expression of the means and variances follows from f_i being the distribution of $\boldsymbol{x}_i^T \boldsymbol{y}_a$. Note that this approximation does not require the embedding vectors to be, themselves, drawn from a multivariate Gaussian distribution. As a remark, we move from an integral on $[-h, h]$ to one over the real axis, but the contribution from the tails is negligible since the integrand goes to zero at least as fast as $e^{-|t|}$. Combining the mixture of Gaussians approximation with Theorems 2.1, 2.2, we can formulate our main result. For $\kappa \geq 1$ and a set of parameters $\{\pi_a, \boldsymbol{\mu}_a, \Omega_a\}_{a=1,\dots,\kappa}$ one can write

$$\frac{Z_i}{m} \approx \sum_{\alpha=1}^{\kappa} \pi_\alpha\ e^{\boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha + \frac{1}{2} \boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i}\ . \tag{2}$$

The remarkable advantage of Equation (2) is that – given the parameters $\{\pi_\alpha, \boldsymbol{\mu}_\alpha, \Omega_\alpha\}_{\alpha=1,\dots,\kappa}$ – Z_i is computed in $\mathcal{O}(\kappa d^2)$ operations, independently of m. This allows us to estimate all Z_i's in $\mathcal{O}(n)$ operations, provided that the parameters can also be obtained in $\mathcal{O}(n)$ operations. In the case $\kappa = 1$, the parameters $\boldsymbol{\mu}, \Omega$ are the mean vector and covariance matrix of the embedding and $\pi = 1$. These parameters are indeed obtained in $\mathcal{O}(m) = \mathcal{O}(n)$ operations. For $\kappa > 1$ we split the dataset with a clustering algorithm, defining a labeling function $\ell : \mathcal{V} \to \{1, \dots, \kappa\}$. We define $\boldsymbol{\mu}_\alpha, \Omega_\alpha$ as the mean vector and covariance matrix of the embedding vectors of class α, and π_α as its relative size. Letting $\mathcal{V}_\alpha = \{i \in \mathcal{V}\ : \ell_i = \alpha\}$,

$$\pi_\alpha = \frac{|\mathcal{V}_\alpha|}{m}; \quad \boldsymbol{\mu}_\alpha = \frac{1}{|\mathcal{V}_\alpha|} \sum_{i\in\mathcal{V}_\alpha} \boldsymbol{x}_i; \quad \Omega_\alpha = \frac{1}{|\mathcal{V}_\alpha| - 1} \sum_{i\in\mathcal{V}_\alpha} (\boldsymbol{x}_i - \boldsymbol{\mu}_\alpha)(\boldsymbol{x}_i - \boldsymbol{\mu}_\alpha)^T\ . \tag{3}$$

This step can still be performed in $\mathcal{O}(n)$ operations, thus allowing the linear-time estimation of all the Z_i's in linear time with respect to n. We now proceed by testing our approximation on pre-trained embedding vectors and we confront the results with comparable methods.

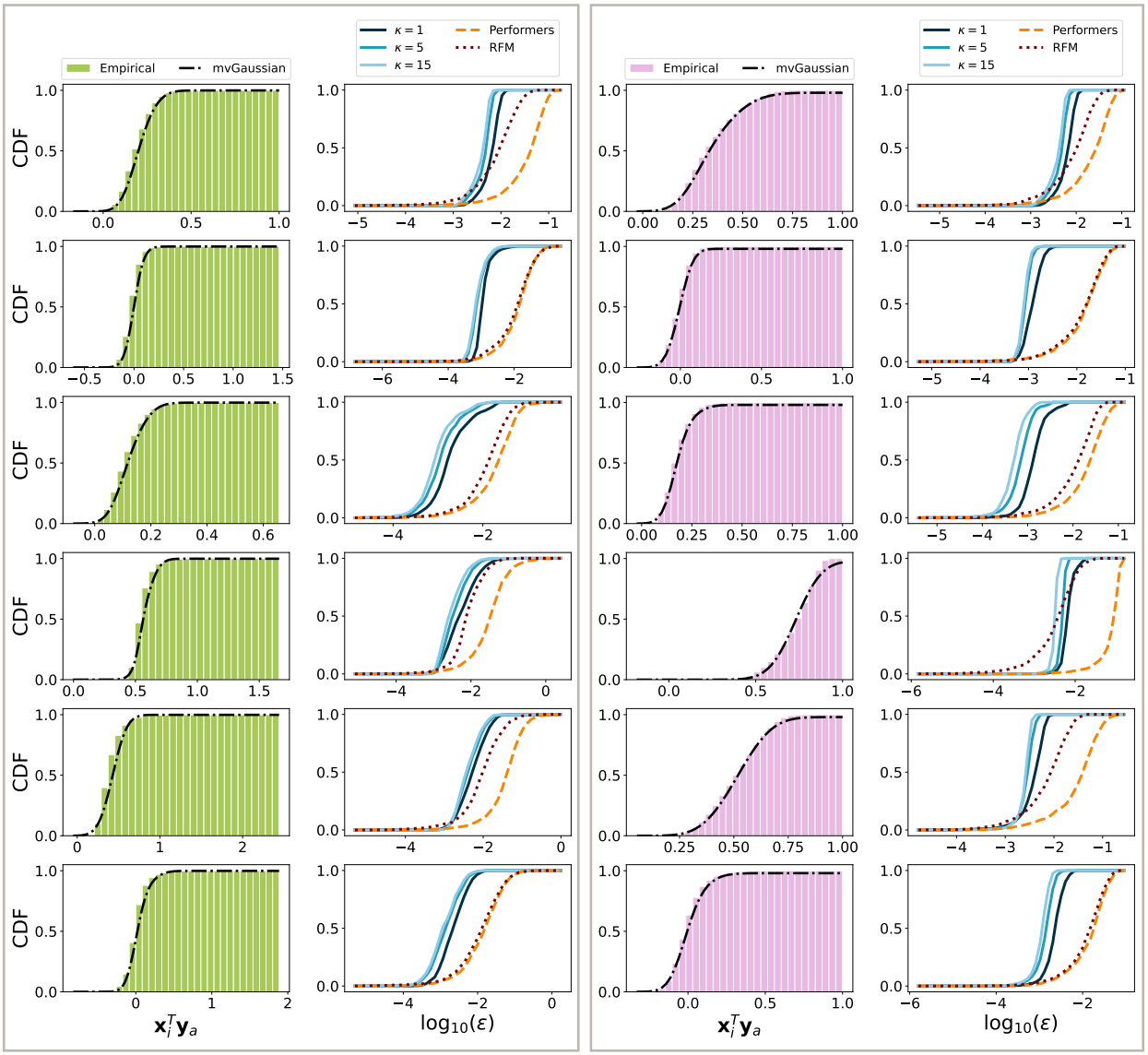

Figure 1: **Empirical evaluation of the attention score normalization constants.** Each row corresponds to one of the six embeddings listed in Section 2.2 which appear in the same order. The first two columns consider the rescaled embedding matrices in which the average norms are equal to one. In the last two, all embedding vectors are normalized to one. The first and third columns show the CDF of the scalar product distribution f_i for an arbitrary index i. The black dash-dotted curve is the estimated multi-variate Gaussian approximation $\tilde{f}_i$ for $\kappa = 5$. The second and fourth columns show the CDF of the relative error ϵ in estimating the attention score normalization constants. The solid lines are referred to our proposed method for three values of κ (color-coded). The dashed line is the Performer method of Choromanski et al. (2020), while the dotted line is the Random Feature Attention of Peng et al. (2021).

2.2 Empirical evaluation

We consider 6 datasets taken from the NLPL word embeddings repository[2] (Kutuzov et al., 2017), representing word embeddings obtained with different algorithms and trained on different corpora:

0. *British National Corpus*; Continuous Skip-Gram, $n = 163.473$, $d = 300$;
7. *English Wikipedia Dump 02/2017*; Global Vectors, $n = 273.930$, $d = 300$;
16. *Gigaword 5th Edition*; fastText Skipgram, $n = 292.967$, $d = 300$;

[2]The datasets can be found at http://vectors.nlpl.eu/repository/ and are shared under the CC BY 4.0 license.

30. *Ancient Greek CoNLL17 corpus*; Word2Vec Continuous Skip-gram, $n = 45742$, $d = 100$;
187. *Taiga corpus*; fastText Continuous Bag-of-Words, 192.415, $d = 300$;
224. *Ukrainian CoNLL17 corpus*; Continuous Bag-of-Word, $n = 99.884$, $d = 200$.

The number reported in the list above, corresponds to the ID used in the repository. For each dataset we (1) re-scale the embedding vectors so that their average norm equals 1; (2) sample 1000 random indices; (3) compute the corresponding exact and the estimated Z_i values for different approximation orders κ. We then repeat the same procedure by imposing that all embedding vectors have unitary norms. Figure 1 shows the results of this procedure. The first column compares the empirical distribution cumulative density functions (CDF) of f_i and $\tilde{f}_i$ obtained for $\kappa = 5$. Here, i is a randomly selected node and $X = Y$ and we use the rescaled version of the embedding matrix. The third column shows the same result for the normalized embedding matrices. The plots confirm that in all cases, the multivariate Gaussian approximation we introduced achieves high accuracy in estimating f_i. We compare our estimation method with two strategies based on the kernel trick Choromanski et al. (2020); Peng et al. (2021). According to these methods, one first defines a random matrix $W \in \mathbb{R}^{D\times d}$ with entries distributed according to $\mathcal{N}(\mathbf{0}_d, I_d)$, then defines a mapping, $\phi_W : \mathbb{R}^d \to \mathbb{R}^{2D}$ so that $\mathbb{E}_W[(\phi_W(\boldsymbol{x}))^T \phi_W(\boldsymbol{y})] = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2}}$. This approximation allows one to efficiently compute Z_i:

$$Z_i = \sum_{a=1}^{m} e^{\boldsymbol{x}_i^T \boldsymbol{y}_a} = e^{\frac{\|\boldsymbol{x}_i\|^2}{2}} \sum_{a=1}^{m} e^{\frac{\|\boldsymbol{y}_a\|^2}{2}}\ \mathbb{E}\left[(\phi_W(\boldsymbol{x}_i))^T \phi_W(\boldsymbol{y}_a)\right] = e^{\frac{\|\boldsymbol{x}_i\|^2}{2}}\ \mathbb{E}\left[(\phi_W(\boldsymbol{x}_i))^T \boldsymbol{m}_W\right] \ ,$$

where $\boldsymbol{m}_W = \sum_{a=1}^{k} e^{\frac{\|\boldsymbol{y}_a\|^2}{2}} \phi_W(\boldsymbol{y}_a)$ needs to be computed only once. The second column of Figure 1 shows the CDF of the relative error ϵ (in logarithmic scale) for our proposed approach with three values of κ and the methods of Choromanski et al. (2020); Peng et al. (2021). The fourth column of Figure 1 shows the results of the same procedure applied to the normalized embedding vectors. As expected, the precision increases with κ, but low errors are obtained for $\kappa = 1$, already. In this case, the clustering step can be omitted and is, therefore, of particular interest. Moreover, even for $\kappa = 1$, we systematically obtain better results than Choromanski et al. (2020); Peng et al. (2021) by even orders of magnitude. It has to be noted that the accuracy of these methods increases by increasing the dimension of the projection, D. Here we choose $D = 1000$ for which we have comparable computation times with our method.

3 `EDRep`: an algorithm for efficient distributed representations

Building on the results of Section 2, we describe an efficient algorithm to obtain distributed representations. We name this algorithm `EDRep` and formulate it as a general embedding problem. One of the main advantages of this formulation is that it can be easily customized to an arbitrary setting. In fact, given a set of entities $\mathcal{V}$, the embedding algorithm requires the description of a probability distribution relating pairs of elements in $\mathcal{V}$ and translates it into an embedding. The optimal design of the probabilities is a problem-dependent task, which is beyond the scope of this article. We show in a few applications that simple choices lead to comparable results with competing or better `2Vec` algorithms but with a lower computation time.

3.1 Problem formulation

We consider a set $\mathcal{V}$ of n items to be embedded. For all $i \in \mathcal{V}$, 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 aim at giving a distributed representation of $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$, in the form of an embedding matrix $X \in \mathbb{R}^{n\times d}$ preserving the relational patterns encoded by the probability distributions. We adopt a variational approach, approximating $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$ with a distribution parameterized by the embedding vectors. To this, we add the regularization term to improve the embedding quality. We define X as the solution to the following optimization problem:

$$X = \underset{Y\in\mathcal{U}_{n\times d}}{\arg\min} \sum_{i\in\mathcal{V}} \Bigg[\underbrace{-\sum_{j\in\mathcal{V}} p_j^{(i)} \log\Big(\mathrm{SoftMax}(YY^T)_{ij}\Big)}_{\text{cross-entropy}} + \underbrace{\boldsymbol{y}_i^T \sum_{j\in\mathcal{V}} p_{0,j}\ \boldsymbol{y}_j}_{\text{regularization}} \Bigg] \ , \tag{4}$$

Algorithm 1 `EDRep`

Input: $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$, set of probabilities; $\boldsymbol{p}_0 \in \mathbb{R}^n$ vector of regularization weights; 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
$X \leftarrow$ initialize the embedding matrix with random unitary vectors
$\eta \leftarrow \eta_0$ initial learning rate
$\pi_{\alpha=1,\dots,\kappa} \leftarrow$ as per Equation (3)
for $1 \le t \le$ `n_epochs` **do**
  $\boldsymbol{\mu}_{\alpha=1,\dots,\kappa}, \Omega_{\alpha=1,\dots,\kappa} \leftarrow$ update the parameters as per Equation (3)
  $\{\boldsymbol{g}_i\}_{i\in\mathcal{V}} \leftarrow$ 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 for
  $\eta \leftarrow \eta - \frac{\eta_0}{\text{n\_epochs}}$ linear update of the learning rate
end for

where $\mathcal{U}_{n\times d}$ denotes the set of all matrices of size $n \times d$ having in their rows unitary vectors and $\boldsymbol{p}_0 \in \mathbb{R}^n$ is so that $\boldsymbol{p}_0^T \mathbf{1}_n = 1$. This cost function is the combination of a cross-entropy with a regularization term. To understand the role of $\boldsymbol{p}_0$, consider for instance $p_{0,i} = n^{-1}$ for all i and let $\boldsymbol{\mu}$ be the mean embedding vector. In this case, the regularization is proportional to $\|\boldsymbol{\mu}\|^2$, thus promoting centered embedding vectors. With other choices of $\boldsymbol{p}_0$, one can give a different weight to each item.

The computational bottleneck of this optimization problem lies in the calculation of the attention score normalization constants. In fact, we let $P \in \mathbb{R}^{n\times n}$ be the matrix with entries $P_{ij} = p_j^{(i)}$ and denote with E the number of non-zero entries of P. We further denote with $\mathbf{1}_n$ the all-ones vector of size n and with $\mathrm{tr}(\cdot)$ the trace operator. Then, the optimization problem of Equation (4) can be reformulated as follows:

$$X = \underset{Y\in\mathcal{U}_{n\times d}}{\arg\min} \left[\underbrace{-\mathrm{tr}\left(Y^T P Y\right)}_{\mathcal{O}(Ed)} + \underbrace{\sum_{i\in\mathcal{V}} \log(Z_i)}_{\mathcal{O}(dn^2)} + \underbrace{\mathrm{tr}\left(Y^T \mathbf{1}_n \boldsymbol{p}_0 Y^T\right)}_{\mathcal{O}(nd)} \right], \tag{5}$$

where the values below the brackets indicate the computational cost required for each element. The derivation of this expression is reported in Appendix B. In many relevant settings, P is a sparse matrix, thus $E \ll n^2$. The calculation of the attention score normalization constants, instead, requires $\mathcal{O}(dn^2)$ operations regardless of E and is the computational bottleneck. We can thus adopt the approximation introduced in Equation (2) to efficiently optimize this cost function and to define an efficient embedding algorithm.

Remark 3.1. *In Equation (4) and in the remainder, we focus on distributions $\boldsymbol{p}^{(i)}$ that are defined and indexed over the same set $\mathcal{V}$. The optimization problem in Equation (4) can however be generalized to an asymmetric scenario in which the entries $p_a^{(i)}$ are defined for $i \in \mathcal{V}$ and $a \in \mathcal{W}$, thus invoking the term* $\mathrm{SoftMax}(XY^T)$*, for $X \neq Y$. We provide* `Python` *codes to obtain the embedding also in this setting.*

Let us now detail the main steps needed to translate the result of Equation (2) into a practical algorithm to produce efficient distributed representations.

3.2 Optimization strategy

We obtain the embedding matrix X by optimizing the problem formulated in Equation (4) with stochastic gradient descent. We substitute the approximated values of Z_i introduced in Equation (2) and we let $M \in \mathbb{R}^{\kappa\times d}$ have the $\{\boldsymbol{\mu}_\alpha\}_{\alpha=1,\dots,\kappa}$ values in its rows. We further define $\mathcal{Z} \in \mathbb{R}^{n\times\kappa}$ as

$$\mathcal{Z}_{i\alpha} = \pi_\alpha \exp\left\{ \boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha + \frac{1}{2} \boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i \right\}.$$

With this notation $Z_i/n = (\mathcal{Z}\mathbf{1}_\kappa)_i$. The $i \in \mathcal{V}$ and $q \in \{1, \dots, d\}$ gradient component reads

$$g_{iq} = -\underbrace{[(P + P^T)X]_{iq}}_{\mathcal{O}(Ed)} + \underbrace{[\mathbf{1}_n \boldsymbol{p}_0^T X + \boldsymbol{p}_0 \mathbf{1}_n^T X]_{iq}}_{\mathcal{O}(nd)} + \underbrace{\frac{1}{(\mathcal{Z}\mathbf{1}_\kappa)_i}\left[\mathcal{Z}M + \sum_{\alpha=1}^{\kappa} \mathcal{Z}_{i\alpha}(X\Omega_\alpha)\right]_{iq}}_{\mathcal{O}(\kappa n d^2)} , \tag{6}$$

where the values underneath the brackets indicate the computational complexity required to compute each addend of the gradient matrix. The derivation of Equation (6) is reported in Appendix C. To keep the normalization, we first compute $\boldsymbol{g}'$ removing the component parallel to $\boldsymbol{x}_i$, then we normalize it and obtain $\boldsymbol{g}_i''$ to finally update the embedding as follows for $0 \leq \eta \leq 1$: $\boldsymbol{x}_i^{\text{new}} = \sqrt{1-\eta^2}\ \boldsymbol{x}_i - \eta \boldsymbol{g}_i''$, that implies $\|\boldsymbol{x}_i^{\text{new}}\| = 1$. Note that Algorithm 1 requires the labeling vector $\boldsymbol{\ell}$ as an input but it is generally unknown. A workaround consists in running `EDRep` for $\kappa = 1$ for which $\boldsymbol{\ell} = \mathbf{1}_n$, then run κ-class clustering and rerun `EDRep` algorithm for the so-obtained vector $\boldsymbol{\ell}$.

3.3 Computational complexity

To determine the complexity of Algorithm 1, let us focus on its computationally heaviest steps:

1. *The calculation of $\boldsymbol{\ell}$ if $\kappa > 1$.* This is obtained in $\mathcal{O}(n\kappa d)$ operations with *k-means* algorithm.
2. *The parameters update as per Equation* (3). This step is performed in $\mathcal{O}(n\kappa d^2)$ operations.
3. *The gradient calculation as per Equation* (6). This is obtained in $\mathcal{O}(Ed + n\kappa d^2)$ operations as indicated by the brackets in Equation (6).

The gradient calculation is then the most expensive operation. Our approximation reduces the complexity required to compute the "Z part" of the gradient from $\mathcal{O}(dn^2)$ to $\mathcal{O}(\kappa n d^2)$, with $\kappa \ll n$. The most expensive term thus requires $\mathcal{O}(Ed)$ operations. This complexity is prohibitive for dense matrices, but in typical settings P is sparse and the product can be performed efficiently. Nonetheless, even for large values of E, if P can be written as the product (or sum of products) of sparse matrices, PX can still be computed efficiently. In fact, let $P = P_m \cdot P_{m-1} \dots P_1$ for some positive m, then PX can be obtained without materializing P, 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 adopting a method such as the one presented in (Le Magoarou & Gribonval, 2016) to approximate a dense matrix P with the product of sparse matrices to speed up the algorithm.

3.4 Comparison with exact gradient computation

We compare the `EDRep` algorithm described in the previous section with its analogous counterpart in which the gradient of Equation (4) is computed analytically in $\mathcal{O}(n^2 d)$ operations. We considered P as the row-normalized random matrix in which the (ij) entry is set to 1 with probability proportional to $\theta_i\theta_j$ and θ_i follows a negative binomial distribution with parameters $N = 3$, $p = 0.3$. This choice allows us to generate heterogeneity in the P structure while being capable of controlling the size n. Figure 2A shows the computational time corresponding to the exact gradient calculation, while panel B reports the same result for `EDRep`. Let X_t be the `EDRep` embedding at epoch t and Y_t be the corresponding one of the full gradient calculation, we define $C_t = \frac{1}{n}\|X_t X_t^T - Y_t Y_t\|_{\text{F}}$, quantifying the deviation between the two embedding methods. Figure 2C shows the behavior of C_t for different κ values, evidencing only slight disagreements between the exact and the approximated embeddings that, as expected, decrease with κ.

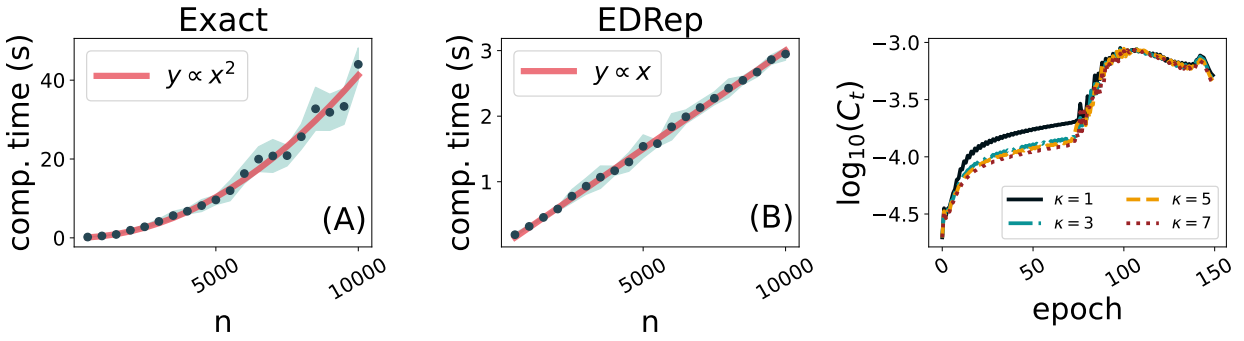

Figure 2: **Comparison with exact gradient calculation**. *Panels A, B*: computation time of the optimization of Equation (4) with gradient descent as function of the size n. Panel A refers to the exact gradient, panel B is Algorithm 1. The blue dots are the mean obtained over 10 realizations, the shadow line has the width of one standard deviation. *Panel C*: logarithm of embedding error $C_t = \frac{1}{n}\|X_t X_t^T - \bar{X}_t \bar{X}_t^T\|_{\mathrm{F}}$ between the true and the estimated embedding matrices at training epoch t. In this experiment, $n = 3000$. In color code and marker style, we report the results for different values of κ. The embedding algorithms are run with the same initial condition and parameters: $\eta_0 = 0.7, d = 32, n_{\mathrm{epochs}} = 25$ (for the first two plots).

4 Use cases

We consider a few use cases of our algorithm to test it and showcase its flexibility in practical settings. To perform these tests, we must specify the set $\{\boldsymbol{p}^{(i)}\}_{i\in\mathcal{V}}$ and we adopt simple strategies to define it. We show that, even for our simple choices, the `EDRep` approach achieves competitive (sometimes superior) results in terms of performance with competing `2Vec` algorithms, with a (much) lower computational time.[3] We would like to underline that the sampling probabilities choice is a hard and problem-dependent task and optimally addressing it is beyond the scope of this article. Our aim is not to develop state-of-the-art algorithms for specific problems but to show that with simple choices we can adapt our algorithm to compete with the closest competing methods in terms of speed and accuracy. For further implementation details regarding the next section, we refer the reader to Appendix D.

4.1 Community detection

Graphs are mathematical objects that model complex relations between pairs of items. They are formed by a set of n *nodes* $\mathcal{V}$ and a set of *edges* $\mathcal{E}$ connecting node pairs (Newman, 2003). Graphs can be represented 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. A relevant problem in graph learning is *community detection*, the task of determining a non-overlapping node partition, unveiling more densely connected groups of nodes (Fortunato & Hric, 2016). A common way of proceeding – see *e.g.* (Von Luxburg, 2007) – is to create a node embedding encoding the community structure and then clustering the nodes in the embedded space. Following this strategy, we adopt the `EDRep` to produce a node embedding with P being $P = \frac{1}{w}\sum_{t=1}^{w} (L_{\mathrm{rw}})^t$, where L_{rw} is the *row-normalized adjacency matrix*. 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 $\boldsymbol{p}_0 = \mathbf{1}_n/n$.

We evaluate our algorithm on synthetic graphs generated from the *degree-corrected stochastic block model* (DCSBM) (Karrer & Newman, 2011), capable of creating graphs with a community structure and an arbitrary degree distribution.[4] The inference accuracy is expressed with the normalized mutual information between the inferred and the ground truth partition. This score ranges between 0 (random assignment) and 1 (perfect assignment). Figure 3A shows the NMI for different values of α, a function of the generative model parameters, controlling for the hardness of the reconstruction problem.[5] The results are compared against other two algorithms that were alternatively deployed to obtain the embedding: the spectral method of

[3]It should be noted that performances typically increase with training time The parameter choice of the `2Vec` algorithms is such that the competing algorithms are comparable on one of the two measures so that the other can be evenly compared.

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

[5]Detection is theoretically feasible if and only if a function of the model parameters $\alpha > 1$ (Gulikers et al., 2018; 2017).

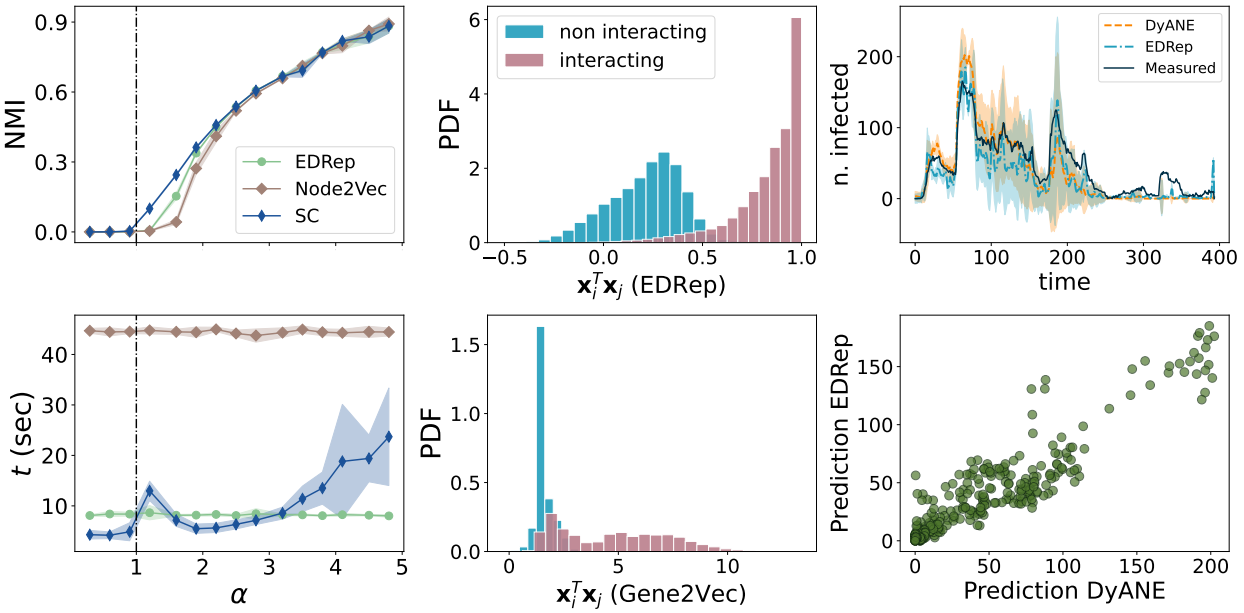

Figure 3: **`EDRep` use cases**. *First column: community detection. Panel A*: normalized mutual information (NMI) as a function of the problem hardness α (see Appendix D) for a DCSBM graph. We consider graphs with $n = 30.000$ nodes, expected average degree $c = 10$, and $q = 4$ communities. The green circles refer to the `EDRep` algorithm with $d = 32$, $\kappa = 1$ and $w = 3$, the brown diamonds are `DeepWalk` with $d = 32$, while the blue narrow diamonds are the spectral clustering algorithm of (Dall'Amico et al., 2021). *Panel B*: corresponding computational time in seconds. *Panels A and B*: averages are over 10 samples and the error bar width equals the standard deviation. *Second column: gene embedding.* Scalar product between gene embedding representation for non-interacting (blue histogram) and interacting (purple histogram) gene pairs. *Panel C*: embedding obtained with `EDRep`; *Panel D* embedding obtained with the `Gene2Vec` algorithm. *Third column: dynamic aware node embedding. Panel E*: number of infected individuals of a SIR process on a proximity network (black solid line) and reconstructed values by `DyANE` (orange dashed line) and using the `EDRep` embedding (blue dashed-dotted line), with $d = 200$ for both methods. The shaded lines are the standard deviations over 50 random trials of the process. *Panel F*: with reference to Panel E, this is the scatter plot between the predicted number of infected per time-stamp by the two strategies.

(Dall'Amico et al., 2021) that was shown to be nearly Bayes-optimal for this task and `DeepWalk` (Perozzi et al., 2014).[6] The communities are obtained from the embeddings using k-class *k-means* clustering.

The results show that the `EDRep`-based algorithm performs almost as well as the optimal algorithm of (Dall'Amico et al., 2021) and a slight mismatch is only observed for α approaching 1. This is a particularly challenging setting in which only a few algorithms can retrieve the community structure. Compared to the `DeepWalk` approach, our method generally yields better results for all α. Figure 3B further compares the computation times, giving the `EDRep` approach a decisive advantage with respect to `DeepWalk`. The main advantage with respect to the spectral algorithm, instead, is the algorithm's computational complexity. For a graph with q communities, the considered spectral clustering algorithm runs in $\mathcal{O}(nq^3)$ operations, while the complexity of `EDRep` is independent of q.

4.2 Gene embeddings

In (Du et al., 2019), the authors develop an algorithm to embed DNA genes from a list of pairs whose co-expression exceeds a threshold value. The dataset comprises 8832 genes and 263016 gene pairs and a list of gene pairs with a binary label indicating whether or not that corresponds to an interacting pair. The `Gene2Vec` algorithm of (Du et al., 2019) builds on `Word2Vec` to obtain meaningful gene vector representations based on their co-expression and uses it to predict pairs of interacting genes.

[6]For the spectral method we used the authors' `Python` implementation available at lorenzodallamico.github.io/codes under the CC BY 4.0 license. For `DeepWalk`, we used the `C++` implementation of github.com/thibaudmartinez/node2vec with its default values, released under the Apache License 2.0.

We obtain the `EDRep` embedding using the row-normalized gene co-occurrence matrix as our choice of P. The `Gene2Vec` embedding is generated with the code provided by the authors with default parameters. Both embeddings have dimension $d = 200$. The computation time of `EDRep` is approximately 6 seconds against the 12 seconds needed for `Gene2Vec`. We then train a logistic regression classifier on the embedding cosine similarities with the 70% of the labeled data and test it on the remaining 30% of the data. Our model achieves an accuracy of 92% and outperforms `Gene2Vec` which has an accuracy of 84%. Figures 3(C, D) show the histogram of the cosine similarities between interacting and non-interacting groups that visually explains the performance gap.

4.3 Causality aware temporal graph embeddings

In (Sato et al., 2021) the authors describe a method to embed temporal networks while preserving the role of time in defining causality. Temporal networks are represented as a sequence of temporal edges (i, j, t), denoting an interaction between i and j at time t. The method relies on the embedding of a *supra-adjacency* matrix, A_{supra} in $\mathbb{R}^{D\times D}$, where $D = \sum_{t=1}^{T} |\mathcal{V}_t|$ and $\mathcal{V}_t$ is the set of active nodes at time t. Here, each node corresponds to a pair "node-time" in the original temporal graph. The *supra-adjacency* matrix is the adjacency matrix of a weighted directed acyclic graph, accounting for time-driven causality. In (Sato et al., 2021) the authors use `DeepWalk` to obtain an embedding from A_{supra} and use it to reconstruct the states of a partially observed dynamical process taking place on the temporal graph (such as an epidemic spreading) from few observations. Following the same procedure of (Sato et al., 2021), we obtain Figure 3(e-f), in which we compare the reconstruction of an epidemic spreading obtained using `DeepWalk` against `EDRep`, with P being the row-normalization of A_{supra}. The results are barely distinguishable, but `EDRep` is more than 5 times faster than the competing approach.

5 Conclusions

In this work, we introduced a computationally efficient method to estimate the normalization constants of attention scores for embedding vectors with bounded norms. Our main result relies on two theorems that characterize concentration properties of the normalization constants and that are formulated under very general assumptions. From these theoretical results, we introduced a heuristic approximation and obtained an explicit formula to compute the attention score normalization in linear time with the problem size. From this result, we then described an embedding algorithm based on the optimization of attention scores. This sort of loss function commonly appears in many embedding algorithms of the `2Vec`-type, but it is not optimized because of its computational complexity. The most common strategy, in fact, is considering a different cost function with linear complexity in the problem size.

We tested our approximation on several empirical data, showing that it can achieve high accuracy levels, outperforming competing methods based on the kernel trick. We then tested our embedding algorithm on a few use cases. The `EDRep` embedding algorithm we defined is very general and problem-agnostic. It takes as input a probability matrix P that encodes relational patterns among the embedded objects. To benchmark the results against comparable methods, we had to make specific choices for the matrix P. The simulations showed that simple and intuitive definitions of such matrix could lead to higher or similar performances compared with competing algorithms. We also observed the `EDRep` algorithm to be systematically faster than its competitors. Despite its simplicity, we believe that the practicality of this algorithm can make it particularly appealing to machine learning users who only need to define the proper P for their problem. We further note that, given the generality of our theoretical argument, one can effortlessly adapt `EDRep` to similar cost functions. The most immediate changes can be done by considering a contrastive learning setting in which also the term $\text{SoftMax}(-XY^T)$ appears. With minor modifications to Algorithm 1, one can also consider non-normalized (but bounded) embedding vectors, or choose P as an arbitrary non-negative matrix. This flexibility enriches the practicality of the proposed algorithm.

Let us now consider some limitations of our work. Given the generality of the formulation, we did not provide a bound to the error on the Z_i estimation. However, we extensively tested our method on several embeddings and matrices P, weighted and unweighted, symmetric and not, and with different sparsity levels. In all cases, the results confirmed the goodness of our proposed approach. The reasons justifying these results are two: (1)

our approximation needs only to hold in "distribution" sense and we do not need a more stringent point-wise accuracy; (2) the multivariate normal approximation is particularly powerful to approximate the distribution of the scalar product. Unsurprisingly, we also observed that the performance of `EDRep` – compared to the `2Vec` methods – highly depends on the matrix P. In some cases, our method provides a neat advantage in terms of performance, while in others the results are essentially identical, with our method being faster. We lack a clear interpretation of the role of P in determining the embedding quality and the convergence speed and we believe this aspect deserves further investigation in the future. We highlight, however, that this analysis is task-dependent, and finding a good P should specifically address a precise research question.

References

R Bamler and S Mandt. Extreme classification via adversarial softmax approximation. In *International Conference on Learning Representations*, 2020.

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

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

Yoshua Bengio and Jean-Sébastien Senécal. Quick training of probabilistic neural nets by importance sampling. In *International Workshop on Artificial Intelligence and Statistics*, pp. 17–24. PMLR, 2003.

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.

Ciro Cattuto, Wouter Van den Broeck, Alain Barrat, Vittoria Colizza, Jean-François Pinton, and Alessandro Vespignani. Dynamics of person-to-person interactions from distributed rfid sensor networks. *PLOS ONE*, 5(7):e11596, 07 2010. doi: 10.1371/journal.pone.0011596. URL http://dx.doi.org/10.1371%2Fjournal.pone.0011596.

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*, pp. 99–107, 2018.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. In *International Conference on Learning Representations*, 2020.

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*, pp. 4039–4049, 2019.

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.

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)*, pp. 2942–2946. IEEE, 2019.

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.

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

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.

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.

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

Matt J Keeling and Pejman Rohani. *Modeling infectious diseases in humans and animals*. Princeton university press, 2011.

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*, pp. 271–276. Linköping University Electronic Press, 2017.

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

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.

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

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

Daniel McFadden. Modelling the choice of residential location. *Transportation Research Record*, 1977. URL https://api.semanticscholar.org/CorpusID:9864447.

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.

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

Cun Mu, Guang Yang, and Yan Zheng. Revisiting skip-gram negative sampling model with rectification. In *Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 1*, pp. 485–497. Springer, 2019.

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

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

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*, pp. 701–710, 2014.

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

Ankit Singh Rawat, Jiecao Chen, Felix Xinnan X Yu, Ananda Theertha Suresh, and Sanjiv Kumar. Sampled softmax with random fourier features. *Advances in Neural Information Processing Systems*, 32, 2019.

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.

Yingchun Shan, Chenyang Bu, Xiaojian Liu, Shengwei Ji, and Lei Li. Confidence-aware negative sampling method for noisy knowledge graph embedding. In *2018 IEEE International Conference on Big Knowledge (ICBK)*, pp. 33–40. IEEE, 2018.

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

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 14138–14148, 2021.

A Proofs

In this appendix we provide the proofs of Theorem 2.1 and 2.2, which we also report here for convenience.

Theorem (2.1)**.** *Consider a vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and a set $\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_m\}$ of m independent random vectors in $\mathbb{R}^d$. Let $|\boldsymbol{x}_i^T \boldsymbol{y}_a| \leq h = \mathcal{O}_m(1)$ for all a. Letting Z_i be defined as in Equation (1), then for all $t > 0$*

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

Before proceeding with the proof of Theorem 2.1, let us first enunciate the following concentration theorem that we will use in our demonstration.

Theorem A.1 ((Talagrand, 1995), (Ledoux, 2001), Corollary 4.10)**.** *Given a random vector $\boldsymbol{\omega} \in [u, v]^n$ 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:*

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

Proof (Theorem 2.1)**.** *Let $\boldsymbol{\omega}^{(i)} \in \mathbb{R}^m$ be a vector with entries $\{\boldsymbol{x}_i^T \boldsymbol{y}_a\}_{a=1,\dots,m}$. Then, the vector $\boldsymbol{\omega}^{(i)}$ satisfies the hypothesis of Theorem A.1 for all i and for $v = -u = h$. We let g be:*

$$g(\boldsymbol{w}^{(i)}) = \frac{1}{m}\sum_{a=1}^{m} e^{w_a^{(i)}} \ ,$$

then one can immediately verify that $Z_i = mg(\boldsymbol{\omega}^{(i)})$. We are left to prove that g satisfies the hypotheses of Theorem A.1 as well. Firstly, g is convex because it is the sum of convex functions. We now compute the Lipschitz parameter. Considering $\boldsymbol{\omega}, \boldsymbol{\omega}' \in [-h, h]^m$ one can give the following bound:

$$|g(\boldsymbol{\omega}) - g(\boldsymbol{\omega}')| \overset{(a)}{\leq} \frac{1}{m}\sum_{a=1}^{m}\left|e^{\omega_a} - e^{\omega'_a}\right| \overset{(b)}{\leq} \frac{e}{m}\sum_{a=1}^{m}|\omega_a - \omega'_a| \overset{(c)}{=} \frac{e}{m}\cdot \mathbf{1}_m^T|\boldsymbol{\omega} - \boldsymbol{\omega}'|$$
$$\overset{(d)}{\leq} \frac{e}{m}\cdot\|\mathbf{1}_m\|\cdot\|\boldsymbol{\omega} - \boldsymbol{\omega}'\| = \frac{e}{\sqrt{m}}\|\boldsymbol{\omega} - \boldsymbol{\omega}'\| \ ,$$

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* $\frac{\sqrt{m}g}{e}$ *is* 1*-Lipschitz and is a suitable choice to apply Theorem A.1. Theorem 2.1 is then easily obtained from a small play on* t *and exploiting the relation between* Z_i *and* $g(\boldsymbol{\omega}^{(i)})$*.*

We now proceed with the proof of Theorem 2.2 which we here recall for convenience.

Theorem (2.2)**.** *Under the same assumptions of Theorem 2.1, assume that for all* a*,* $\boldsymbol{x}_i^T \boldsymbol{y}_a$ *converges in distribution to a variable* $\omega_i \sim f_i$*, then,*

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

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 A.2 ((Luxemburg, 1971))**.** *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 (Theorem 2.2)**.** *The values* $\boldsymbol{x}_i^T \boldsymbol{y}_a := \omega_a^{(i)}$ *are a sequence of* m *random variables with cumulative densities* $F_{i,1}, \dots, F_{i,m}$*.*

$$\lim_{m\to\infty} \mathbb{E}\left[\frac{Z_i}{m}\right] = \lim_{m\to\infty} \frac{1}{m}\sum_{a=1}^{m} \mathbb{E}\left[e^{\omega_a^{(i)}}\right] = \lim_{m\to\infty} \mathbb{E}\left[e^{\omega_m^{(i)}}\right] \overset{(a)}{=} \lim_{m\to\infty} \int_{-h}^{h} dt\ e^t f_{i,m}(t)$$

$$\overset{(b)}{=} \lim_{m\to\infty} \left[e^t F_{i,m}(t)\Big|_{-h}^{h} - \int_{-h}^{h} dt\ e^t F_{i,m}(t)\right] \xrightarrow[m\to\infty]{(c)} e^t F_i(t)\Big|_{-h}^{h} - \int_{-h}^{h} dt\ e^t F_i(t) = \int_{-h}^{h} dt\ e^t f_i(t)\ .$$

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 A.2 for* $M = e^h$*.*

B Derivation of Equation (5)

We here report the derivation of Equation (5) from (4).

$$\begin{aligned}
\mathcal{L} &= -\sum_{i,j\in\mathcal{V}} p_j^{(i)} \log\Big(\text{SoftMax}(YY^T)_{ij}\Big) + \sum_{i,j\in\mathcal{V}} p_{0,j}\ \boldsymbol{y}_i^T \boldsymbol{y}_j \\
&\overset{(a)}{=} -\sum_{i,j\in\mathcal{V}} p_j^{(i)} (YY^T)_{ij} + \sum_{i,j\in\mathcal{V}} p_j^{(i)}\ \log(Z_i) + \sum_{i,j\in\mathcal{V}} p_{0,j} (YY^T)_{ij} \\
&\overset{(b)}{=} -\sum_{i,j\in\mathcal{V}} P_{ij} (YY^T)_{ij} + \sum_{i\in\mathcal{V}} \log(Z_i) + \sum_{i,j\in\mathcal{V}} (\mathbf{1}_n \boldsymbol{p}_0^T)_{ij} (YY^T)_{ij} \\
&\overset{(c)}{=} -\sum_{i\in\mathcal{V}} (PYY^T)_{ii} + \sum_{i\in\mathcal{V}} \log(Z_i) + \sum_{i\in\mathcal{V}} (\mathbf{1}_n \boldsymbol{p}_0^T YY^T)_{ii} \\
&\overset{(d)}{=} -\text{tr}(PYY^T) + \sum_{i\in\mathcal{V}} \log(Z_i) + \text{tr}(\mathbf{1}_n \boldsymbol{p}_0^T YY^T) \\
&\overset{(e)}{=} -\text{tr}(Y^T PY) + \sum_{i\in\mathcal{V}} \log(Z_i) + \text{tr}(Y^T \mathbf{1}_n \boldsymbol{p}_0^T Y)\ ,
\end{aligned}$$

where in (a) we used the softmax definition and rewrote $\boldsymbol{y}_i^T \boldsymbol{y}_j = (YY^T)_{ij}$; in (b) we used the definition $P_{ij} = p_j^{(i)}$ and exploited the property $\sum_{j\in\mathcal{V}} p_j^{(i)} = 1$; in (c) we used the fact that YY^T is a symmetric matrix; in (d) we leverage the trace definition; in (e) we use the property of the trace $\text{tr}(AB) = \text{tr}(BA)$.

C Derivation of the gradient

We here derive the gradient expression as it appears in Equation (6). Note that in this derivation, the quantities $\boldsymbol{\mu}_\alpha$, Ω_α are considered as constants, in a stochastic gradient descent fashion. We observed that this gradient form achieves better results in fewer epoch. Let us first rewrite the loss function of Equation (4). Following the passages detailed in Appendix B, we obtain

$$\mathcal{L} = -\sum_{i,j\in\mathcal{V}} (P_{ij} - p_{0,j})\, \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i\in\mathcal{V}} \log(Z_i)\ .$$

We now introduce the approximation of Equation (2) and rewrite

$$\begin{aligned}\mathcal{L} &\approx -\sum_{i,j\in\mathcal{V}} (P_{ij} - p_{0,j})\, \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i\in\mathcal{V}} \log \sum_{\alpha=1}^{\kappa} n\pi_\alpha \exp\left\{\boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha + \frac{1}{2}\boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i\right\} \\ &= -\sum_{i,j\in\mathcal{V}} \sum_{q=1}^{d} (P_{ij} - p_{0,j})\, x_{iq} x_{jq} + \sum_{i\in\mathcal{V}} \log \sum_{\alpha=1}^{\kappa} n\pi_\alpha \exp\left\{\sum_{q=1}^{d} x_{iq}\mu_{\alpha,q} + \frac{1}{2}\sum_{q,p=1}^{d} x_{iq}\Omega_{\alpha,qp} x_{ip}\right\}\ .\end{aligned}$$

We now proceed computing the derivative with respect to x_{kr} to obtain the respective gradient term.

$$\begin{aligned}\partial_{x_{kr}}\mathcal{L} \approx &-\sum_{i,j\in\mathcal{V}} \sum_{q=1}^{d} (P_{ij} - p_{0,j})\, \delta_{qr}\left[\delta_{ik} x_{jq} + \delta_{jk} x_{iq}\right] \\ &+ \sum_{i\in\mathcal{V}} \frac{\sum_{\alpha=1}^{\kappa} \pi_\alpha\ e^{\boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha + \frac{1}{2}\boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i} \left(\sum_{q=1}^{d} \delta_{ik}\delta_{qr}\mu_{\alpha,q} + \sum_{q,p=1}^{d} \Omega_{\alpha,qp}\left[\delta_{ik}\delta_{qr}x_{ip} + \delta_{ik}\delta_{pr}x_{iq}\right]\right)}{\sum_{\alpha=1}^{\kappa} \pi_\alpha\ e^{\boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha + \frac{1}{2}\boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i}}\ .\end{aligned}$$

We now use the notation $\mathcal{Z}_{i\alpha} = \pi_\alpha\ e^{\boldsymbol{x}_i^T \boldsymbol{\mu}_\alpha + \frac{1}{2}\boldsymbol{x}_i^T \Omega_\alpha \boldsymbol{x}_i}$ and compute the sums.

$$\begin{aligned}\partial_{x_{kr}}\mathcal{L} \approx &-\sum_{i\in\mathcal{V}} (P_{ik} - p_{0,k}) x_{ir} - \sum_{j\in\mathcal{V}} (P_{kj} - p_{0,j}) x_{jr} \\ &+ \frac{1}{\sum_{\alpha=1}^{\kappa} \mathcal{Z}_{k\alpha}} \cdot \sum_{\alpha=1}^{\kappa} \mathcal{Z}_{k\alpha} \left(\mu_{\alpha,r} + \frac{1}{2}\sum_{q=1}^{d} x_{kq}\Omega_{\alpha,rq} + \frac{1}{2}\sum_{q=1}^{d} x_{kq}\Omega_{\alpha,qr}\right)\end{aligned}$$

Now we recall that $\sum_{\alpha=1}^{\kappa} \mathcal{Z}_{i\alpha} = Z_i/n$ and $M_{\alpha,r} = \mu_{\alpha,r}$ and that $\Omega_\alpha = \Omega_\alpha^T$.

$$\partial_{x_{kr}}\mathcal{L} \approx -\left[\left(P^T + \boldsymbol{p}_0 \mathbf{1}_n^T\right) X\right]_{kr} - \left[\left(P + \mathbf{1}_n \boldsymbol{p}_0^T\right) X\right]_{kr} + \frac{1}{(\mathcal{Z}\mathbf{1}_\kappa)_k} \cdot \left([\mathcal{Z}M]_{kr} + \sum_{\alpha=1}^{\kappa} \mathcal{Z}_{k\alpha}[X\Omega_\alpha]_{kr}\right)\ ,$$

thus obtaining Equation (6).

D Experiment implementation details

We here report some implementation details in the experiments we conducted. This section complements the information in the main text when this is insufficient to reproduce our results.

D.1 Node embeddings

In the experiments we test the node embedding problem for the task of community detection. To do so, we work with synthetic graphs generated from the degree corrected stochastic block model (DCSBM) (Karrer & Newman, 2011) that we here define.

Definition D.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, with $\mathbb{E}[\theta] = 1$ *and finite variance. 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 $\mathbb{E}[d_i] \propto \theta_i$, thus allowing one to model the broad degree distributions typically observed in real networks (Barabási & Albert, 1999). Given this model, the community detection task consists in inferring 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 $\alpha = (c - c_{\text{out}})\sqrt{\frac{\mathbb{E}[\theta^2]}{c}} > 1$. This is the α parameters appearing in the main text. In the simulations the θ_i's are obtained by: i) drawing a random variable from a uniform distribution between 3 and 12; ii) raising it to the power 6; iii) normalize it so that $\mathbb{E}[\theta] = 1$. This leads to a rather broad degree distribution, even if it maintains a finite support.

For all three methods under comparison we obtain the embedding vectors from A and then cluster the nodes into communities by applying *k-means* and supposing that the number of communities q is known. The algorithm of (Dall'Amico et al., 2019; 2021) obtains the embedding by extracting a sequence of eigenvectors from a sequence of parameterized matrices that are automatically learned from the graph and does not require any parametrization. The `DeepWalk` and `EDRep` algorithms generate embeddings in $d = 32$ dimensions. We observed that the results are essentially invariant in a large spectrum of d values for both embedding algorithms.

D.2 Dynamically aware node embeddings

This experiment features three main steps: i) the creation of the supra-adjacency matrix from a temporal network; ii) the creation of an embedding based on this matrix; iii) the reconstruction of a dynamical process taking place on the network. We now describe these steps in detail.

Definition of the supra-adjacency matrix

We consider a temporal graph collected by the SocioPatterns collaboration. This dataset describes face-to-face proximity encounters between people at a conference. The data are recorded with a temporal resolution of 20 seconds and correspond to interactions within a distance of approximately 1.5 meters.[7] The dataset contains approximately 3000 different time-stamps. Following the procedure of (Sato et al., 2021) we aggregate them into $T = 180$ windows of approximately 15 minutes each. We thus obtain a temporal graph that is represented as a sequence of weighted temporal edges (i, j, t, w_{ijt}), indicating that i, j interacting at snapshot t for a cumulative time equal to w_{ijt}. We say that a node is active at time t if it has neighbors at time t. We let $t_{i,a}$ be the time at which node i is active for the a-th time. Given this graph, we then build the supra-adjacency matrix that is defined as follows.

Definition D.2 (Supra-adjacency matrix)**.** *Consider a temporal graph represented as a sequence of weighted temporal edges* (i, j, t, ω_{ijt})*. We define a set of "temporal nodes" given by all the pairs* $i, t_{i,a}$ *where* i *is a node of the temporal graph and* $t_{i,a}$ *is the time of the* a*-th appearance of* i *in the network. We denote this set* $\mathcal{D}$*, formally defined as*

$$\mathcal{D} = \{(i, t_{i,a}) \ : \ i \in \mathcal{V}, \exists \ j \in \mathcal{V} \ : \ (i, j, t_{i,a}) \in \mathcal{E}\},$$

[7] The experiment is described in (Cattuto et al., 2010). The data are shared under the Creative Commons Public Domain Dedication license and can be downloaded at http://www.sociopatterns.org/datasets/sfhh-conference-data-set/.

where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ of temporal edges. The cardinality of this set is $D = |\mathcal{D}|$. We then define a directed graph with $\mathcal{D}$ being the set of nodes. The directed edges are placed between

$$\begin{cases} (i, t_{i,a}) \to (i, t_{i,a+1}) & \text{self connections} \\ (i, t_{i,a}) \to (i, t_{j,b+1}) & \text{if } t_{i,a} = t_{j,b} \text{ and } (i, j, t) \in \mathcal{E} \\ (i, t_{j,b}) \to (i, t_{a,a+1}) & \text{if } t_{i,a} = t_{j,b} \text{ and } (i, j, t) \in \mathcal{E}. \end{cases}$$

The supra-adjacency matrix $A_{\text{supra}} \in \mathbb{R}^{D \times D}$ is the adjacency matrix of the graph we just defined.

Creation of the embedding

Given A_{supra}, the authors generate an embedding with the `DeepWalk` algorithm. Here, the random walker can only follow time-respecting paths, by construction of the matrix. We let the random walks have length equal to 15 steps and deploy also in this case the embedding dimension $d = 30$. For `EDRep` we use the row-normalized version of A_{supra} as P. As a result we obtain an embedding vector for each $(i, t_{i,a})$ pair.

Reconstruction of a partially observed dynamic process

Following (Sato et al., 2021) we consider an epidemic process taking place on the temporal network. We use the SIR model (Keeling & Rohani, 2011) in which infected nodes (I) can make susceptible nodes (S) to transition to the infected state with a probability β if they are in contact. Infected individuals then recover (R) with a probability μ and are unable to infect or get infected. We run the SIR model letting all nodes to be in the S state at the beginning of the simulation and having one infected node. The experiment is run with $\beta = 0.15$ and $\mu = 0.01$ and it outputs the state of each node at all times, i.e. for all $(i, t_{i,a})$.

We then suppose to observe a fraction of these states (every node is expected to only be observe once) and obtain a binary variable for each $(i, t_{i,a})$ indicating whether node i was infected at time $t_{i,a}$. Exploiting the embeddings, we train a logistic regression model to predict the state of the node in each unobserved time and compare the predicted number of infected individuals against the observed ones. We repeat the experiment for 50 different realizations of the observed training set.