Gravity-Inspired Graph Autoencoders
for Directed Link Prediction

Guillaume Salha*
Deezer Research & Development
LIX, École Polytechnique

Stratis Limnios
LIX, École Polytechnique

Romain Hennequin
Deezer Research & Development

Viet Anh Tran
Deezer Research & Development

Michalis Vazirgiannis
LIX, École Polytechnique

ABSTRACT

Graph autoencoders (AE) and variational autoencoders (VAE) recently emerged as powerful node embedding methods. In particular, graph AE and VAE were successfully leveraged to tackle the challenging link prediction problem, aiming at figuring out whether some pairs of nodes from a graph are connected by unobserved edges. However, these models focus on undirected graphs and therefore ignore the potential direction of the link, which is limiting for numerous real-life applications. In this paper, we extend the graph AE and VAE frameworks to address link prediction in directed graphs. We present a new gravity-inspired decoder scheme that can effectively reconstruct directed graphs from a node embedding. We empirically evaluate our method on three different directed link prediction tasks, for which standard graph AE and VAE perform poorly. We achieve competitive results on three real-world graphs, outperforming several popular baselines.

CCS CONCEPTS

• **Information systems → Social networks**; • **Mathematics of computing → Graph algorithms**; • **Computing methodologies → Learning latent representations**.

KEYWORDS

Directed Graphs, Autoencoders, Variational Autoencoders, Graph Representation Learning, Node Embedding, Link Prediction

1 INTRODUCTION

Graphs are useful data structures to efficiently represent relationships among items. Due to the proliferation of graph data [54, 56],

---

*Contact author: research@deezer.com

a large variety of specific problems initiated significant research efforts from the Machine Learning community, aiming at extracting relevant information from such structures. This includes node clustering [33], influence maximization [21], graph generation [45] and link prediction, on which we focus in this paper.

Link prediction consists in inferring the existence of new relationships or still unobserved interactions (i.e. new *edges* in the graph) between pairs of entities (*nodes*) based on observable links and on their properties [29, 52]. This challenging task has been widely studied and successfully applied to several domains. In biological networks, link prediction models were leveraged to predict new interactions between proteins [26]. It is also present in our daily lives, suggesting people we may know but we are still not connected to, in our social networks [18, 29, 52]. Besides, link prediction is closely related to numerous recommendation tasks [4, 28, 58].

Link prediction has been historically addressed through graph mining heuristics, via the construction of similarity indices between nodes, capturing the likelihood of their connection in the graph. The Adamic-Adar and Katz indices [29], reflecting neighborhood structure and node proximity, are notorious examples of such similarity indices. More recently, along with the increasing efforts in extending Deep Learning methods to graph structures [6, 43, 54], these approaches have been outperformed by the node embedding paradigm [16, 46, 56]. In a nutshell, the strategy is to train graph neural networks to represent nodes as vectors in a low-dimensional vector space, namely the *embedding* space. Ideally, in such space nodes with a structural proximity in the graph should be close to each other. Therefore, one can resort to proximity measures such as inner products between vector representations to predict new unobserved links in the underlying graph. In this direction, the graph extensions of autoencoders (AE) [1, 40] and variational autoencoders (VAE) [23, 48] recently appeared as state-of-the-art approaches for link prediction in numerous experimental analyses [25, 38, 41, 47, 51].

However, these models focus on *undirected graphs* and therefore ignore the potential direction of the link. As explained in section 2, a graph autoencoder predicting that node i is connected to node j will also predict that node j is connected to node i, with the same probability. This is limiting for numerous real-life applications, as *directed graphs* are ubiquitous. For instance, web graphs are made up of directed hyperlinks. In social networks such as Twitter, opinion leaders are usually followed by many users, but only few of these connections are reciprocal. Moreover, directed graphs are efficient abstractions in many domains where data are not explicitly structured as graphs. For instance, on music streaming platforms,

the page providing information about an artist will usually display the k most similar artists. Artists similarities can be represented in a graph, in which nodes are artists, connected to their k most similar neighbors. Such graph is definitely directed: indeed, while Bob Marley might be among the most similar artists of a new unknown reggae band, it is unlikely that this band should be presented among Bob Marley's top similar artists in his page.

Directed link prediction has been tackled through graph mining asymmetric measures [13, 44, 55] and, recently, a few attempts at capturing asymmetric proximity when creating node embeddings were proposed [34, 36, 59]. However, the question of how to reconstruct directed graphs from vector space representations to effectively perform directed link prediction remains widely open. In particular, it is unclear how to extend graph AE and graph VAE to directed graphs and to which extent the promising performances of these models on undirected graphs could also be achieved on directed link prediction tasks. We propose to address these questions in this paper, making the following contributions:

- We present a new model to effectively learn node embeddings from directed graphs using the graph AE and VAE frameworks. We draw inspiration from Newton's theory of universal gravitation to introduce a new decoder scheme, able to reconstruct asymmetric relationships from vector space node embeddings.
- We empirically evaluate our approach on three different directed link prediction tasks, for which standard graph AE and VAE perform poorly. We achieve competitive results on three real-world datasets, outperforming popular baselines. To the best of our knowledge, these are the first graph AE/VAE experiments on directed graphs.
- We publicly release our code[1] for these experiments, for reproducibility and easier future usages.

This paper is organized as follows. In Section 2, we recall key concepts related to graph AE and VAE and we explain why these models are not suitable for directed link prediction. In Section 3, we introduce our gravity-inspired method to reconstruct directed graphs using graph AE or VAE, and effectively perform directed link prediction. We present and discuss our experimental analysis in Section 4, and we conclude in Section 5.

2 PRELIMINARIES

In this section, we provide an overview of graph AE, VAE and of their main applications to link prediction. In the following, we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ without self-loops, with $|\mathcal{V}| = n$ nodes and $|\mathcal{E}| = m$ edges that can be directed. We denote by A the adjacency matrix of $\mathcal{G}$, that is either binary or weighted. Moreover, nodes can possibly have features vectors of size f, gathered in an $n \times f$ matrix X. Otherwise, X is the $n \times n$ identity matrix I.

2.1 Graph Autoencoders

Graph autoencoders [25, 51] are a family of models extending autoencoders [1, 40] to graph structures. Their goal is to learn a node embedding, i.e. a low dimensional vector space representation of the nodes. Graph AE are composed of two stacked models:

---
[1]https://github.com/deezer/gravity_graph_autoencoders

- Firstly, an *encoder* model assigns a latent vector z_i of size d, with $d \ll n$, to each node i of the graph. The $n \times d$ matrix Z of all latent vectors z_i is usually the output of a Graph Neural Network (GNN) applied on A and, where appropriate, on X, i.e. we have $Z = \text{GNN}(A, X)$.
- Then, a *decoder* model aims at reconstructing the adjacency matrix A from Z, using another GNN or a simpler alternative. For instance, in [25] and in several extensions of their model [38, 41], decoding is obtained through an inner product between latent vectors, along with a sigmoid activation $\sigma(x) = 1/(1 + e^{-x})$ or, if A is weighted, some more complex thresholding. In other words, the larger the inner product $z_i^T z_j$, the more likely node i and j are connected in the graph according to the model. Denoting $\hat{A}$ the reconstruction of A from the decoder, we have $\hat{A} = \sigma(ZZ^T)$.

The intuition behind autoencoders is the following: if, starting from the latent vectors, the decoder is able to reconstruct an adjacency matrix $\hat{A}$ that is *close* to the original one, then it should mean that these representations preserve some important characteristics of the graph structure. Graph AE are trained by minimizing the reconstruction loss $\|A - \hat{A}\|_F$ of the graph structure [51], with $\| \cdot \|_F$ the Frobenius matrix norm, or alternatively a weighted cross entropy loss [25], by stochastic gradient descent [15].

2.2 Graph Convolutional Networks

Throughout this paper, as [25] and most subsequent works [10, 17, 38, 41], we assume that the GNN encoder is a Graph Convolutional Network (GCN) [24]. In a GCN with L layers, with $L \geq 2$ and $Z = H^{(L)}$, we have:

$$\begin{cases} H^{(0)} = X \\ H^{(l)} = \text{ReLU}(\tilde{A} H^{(l-1)} W^{(l-1)}) \text{ for } l \in \{1, \ldots L-1\} \\ H^{(L)} = \tilde{A} H^{(L-1)} W^{(L-1)}. \end{cases}$$

In the above equation, $\tilde{A}$ denotes some normalized version of A. As undirected graphs were considered in existing models, a usual choice is the symmetric normalization $\tilde{A} = D^{-1/2}(A + I)D^{-1/2}$, where D is the diagonal degree matrix of $A + I$. In a nutshell, for each layer l, we average the feature vectors from $H^{(l-1)}$ of the neighbors of a given node, together with its own feature information (thus the I) and with a ReLU activation: $\text{ReLU}(x) = \max(x, 0)$. Weights matrices $W^{(l)}$ are trained by stochastic gradient descent.

We rely on GCN encoders for three main reasons: 1) consistency with previous efforts on graph AE, 2) capitalization on previous successes of GCN-based graph AE (see subsection 2.4) and, last but not least, 3) for computation efficiency. Indeed, evaluating each layer of a GCN has a linear complexity w.r.t. the number of edges m [24]. Speed-up strategies to improve the training of GCNs were also proposed [8, 53]. Nonetheless, we point out that the method we present in this article is not limited to GCN and would still be valid for any alternative encoder, e.g. for more complex encoders such as ChebNet [9] that sometimes empirically outperform GCN encoders [41].

2.3 Variational Graph Autoencoders

[25] introduced Variational Graph Autoencoders, denoted VGAE, a graph extension of VAE [23]. While sharing the name *autoencoder*, VAE are actually based on quite different mathematical foundations. Specifically, [25] assume a probabilistic model on the graph that involves some latent variables z_i of length $d \ll n$ for each node $i \in \mathcal{V}$. Such vectors are the node representations in a low dimensional embedding space $\mathcal{Z}$. Denoting by Z the $n \times d$ matrix of all latent vectors, authors define the inference model as follows:

$$q(Z|A, X) = \prod_{i=1}^{n} q(z_i|A, X) \text{ where } q(z_i|A, X) = \mathcal{N}(z_i|\mu_i, \text{diag}(\sigma_i^2)).$$

The latent vectors z_i themselves are random samples drawn from the learned distribution, and this inference step is referred to as the *encoding* part of the graph VAE. Parameters of Gaussian distributions are learned using two GCNs. In other words, μ, the matrix of mean vectors μ_i, is defined as $\mu = \text{GCN}_\mu(A, X)$. Likewise, $\log \sigma = \text{GCN}_\sigma(A, X)$.

Then, a generative model attempts to reconstruct A using, as for graph AE, inner products between latent variables:

$$p(A|Z) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(A_{ij}|z_i, z_j) \text{ where } p(A_{ij} = 1|z_i, z_j) = \sigma(z_i^T z_j).$$

As before, $\sigma(\cdot)$ is the sigmoid activation function. This is the *decoding* part of the model. [25] optimize GCN weights by maximizing a tractable variational lower bound (ELBO) of the model's likelihood:

$$\mathcal{L} = \mathbb{E}_{q(Z|A,X)} \Big[\log p(A|Z) \Big] - \mathcal{D}_{KL}(q(Z|A, X)||p(Z)),$$

with a Gaussian prior $p(Z) = \prod_i p(z_i) = \prod_i \mathcal{N}(z_i|0, I)$, using full-batch gradient descent and leveraging the *reparameterization trick* [23]. $\mathcal{D}_{KL}(\cdot, \cdot)$ is the Kullback-Leibler divergence [27].

2.4 Graph AE and VAE for Undirected Link Prediction

In the last three years, graph AE/VAE and their extensions have been successfully leveraged to tackle several challenging tasks, such as node clustering [38, 41, 50], recommendation from bipartite graphs [4, 10] and graph generation, notably biologically plausible molecule generation from graph VAE's generative models [19, 30, 31, 42, 45]. We refer to the aforementioned references for a broader overview of these applications, and focus on link prediction tasks in the remaining of this section.

Link prediction has been the main evaluation task for graph AE and VAE in the seminal work of [25] and in numerous extensions [17, 38, 41, 47]. In a nutshell, authors evaluate the global ability of their models to predict whether some pairs of nodes from an undirected graph are connected by unobserved edges, using the latent space representations of the nodes. More formally, in such setting autoencoders are usually trained on an incomplete version of the graph where a proportion of the edges, say 10%, were randomly removed. Then, a test set is created, gathering these missing edges and the same number of randomly picked pairs of unconnected nodes. Authors evaluate the model's ability to discriminate the true edges (i.e. $A_{ij} = 1$ in the *complete* adjacency matrix) from the fake ones ($A_{ij} = 0$) using the decoding of the latent vectors $\hat{A}_{i,j} =$

$\sigma(z_i^T z_j)$. In other words, they predict that nodes are connected when $\hat{A}_{i,j}$ is larger than some threshold. This is a binary classification task, typically assessed using the *Area Under the Receiver Operating Characteristic (ROC) Curve (AUC)* or the *Average Precision (AP)* scores. For such tasks, graph AE and VAE have been empirically proven to be competitive and often outperforming w.r.t. several popular node embeddings baselines, notably Laplacian eigenmaps [3] and word2vec-like models such as DeepWalk [39], LINE [46] and node2vec [16].

We point out that most of these experiments are focusing to medium-size graphs with up to a few thousand nodes and edges. This is mainly due to the limiting $O(dn^2)$ quadratic time complexity of the inner product decoder, which involves the multiplication of the dense matrices Z and Z^T. However, [41] recently bypassed this scalability issue and introduced a general framework for more scalable graph AE and VAE, leveraging graph degeneracy concepts [32]. They confirmed the competitive performance of graph AE and VAE for large-scale link prediction, based on experiments on undirected graphs with up to millions of nodes and edges.

2.5 Why do these models fail to perform Directed Link Prediction?

At this stage, we recall that all previously mentioned works assume, either explicitly or implicitly, that the input graph is *undirected*. By design, graph AE and VAE are not suitable for directed graphs, since they are ignoring directions when reconstructing the adjacency matrix from the embedding. Indeed, due to the symmetry of the inner product decoder, we have:

$$\hat{A}_{ij} = \sigma(z_i^T z_j) = \sigma(z_j^T z_i) = \hat{A}_{ji}.$$

In other words, if we predict the existence of an edge (i, j) from node i to node j, then we also necessarily predict the existence of the reverse edge (j, i), with the same probability. As a consequence, as we empirically show in section 4, standard graph AE and VAE significantly underperform on link prediction tasks in directed graphs, where relationships are not always reciprocal.

Replacing inner product decoders by an L_p distance in the embedding (e.g. the Euclidean distance, if $p = 2$) or by existing more refined decoder schemes [17] would lead to the same conclusion, since they are also symmetric. Recently, [57] proposed D-VAE, a variational autoencoders for small Directed Acyclic Graphs (DAG) such as neural networks architectures or bayesian networks, focusing on neural architecture search and structure learning. However, the question of how to extend graph AE and VAE to general directed graphs, such as citation networks or web hyperlink networks, where directed link prediction is challenging, remains open.

2.6 On the Source/Target Vectors Paradigm

To conclude these preliminaries, we highlight that, out of the graph AE/VAE frameworks, a few recent node embeddings methods proposed to tackle directed link prediction by actually learning *two latent vectors* for each node. More precisely:

- HOPE, short for *High-Order Proximity preserved Embedding* [36], aims at preserving high-order node proximities and capturing asymmetric transitivity. Nodes are represented by *two* vectors: *source* vectors $z_i^{(s)}$, stack up in an $n \times d$ matrix

$Z^{(s)}$, and *target* vectors $z_i^{(t)}$, gathered in another $n \times d$ matrix $Z^{(t)}$. For a given $n \times n$ similarity matrix S, authors learn these vectors by approximately minimizing $\|S - Z^{(s)}Z^{(t)T}\|_F$ using a generalized SVD. For directed graphs, a usual choice for S is the Katz matrix $S^{\text{Katz}} = \sum_{i=1}^{\infty} \beta^i A^i$, with $S^{\text{Katz}} = (I - \beta A)^{-1}\beta A$ if the parameter $\beta > 0$ is smaller than the spectral radius of A [20]. It computes the number of paths from a node to another one, these paths being exponentially weighted according to their length. For link prediction, one can assess the likelihood of a link from node i to node j using the asymmetric reconstruction $\hat{A}_{ij} = \sigma(z_i^{(s)T} z_j^{(t)})$.

- APP [59] is a scalable *Asymmetric Proximity Preserving* node embedding method, that conserves the Rooted PageRank score [37] for any node pair. APP leverages random walk with restart strategies to learn, as HOPE, a source vector and a target vector for each node. As before we predict that node i is connected to node j from the inner product of source vector i and target vector j, with a sigmoid activation.

One can derive a straightforward extension of this *source/target vectors* paradigm to graph AE and VAE. Indeed, considering GCN encoders returning d-dim latent vectors z_i, with d being even, we can assume that the $d/2$ first dimensions (resp. the $d/2$ last dimensions) of z_i actually correspond to the source (resp. target) vector of node i, i.e. $z_i^{(s)} = z_{i[1:\frac{d}{2}]}$ and $z_i^{(t)} = z_{i[(\frac{d}{2}+1):d]}$. Then, we can replace the symmetric decoder $\hat{A}_{ij} = \hat{A}_{ji} = \sigma(z_i^T z_j)$ by $\hat{A}_{ij} = \sigma(z_i^{(s)T} z_j^{(t)})$ and $\hat{A}_{ji} = \sigma(z_j^{(s)T} z_i^{(t)})$, both in the AE and VAE frameworks, to reconstruct directed links from encoded representations. We refer to this method as *source/target graph AE (or VAE)*.

However, in the following of this paper, we adopt a different approach and we propose to come back to the original idea consisting in learning a *single* node embedding, and therefore represent each node via a single latent vector. Such approach has a stronger interpretability power and, as we later show in the experimental part of this paper, it also significantly outperforms source/target graph AE and VAE on directed link prediction tasks.

3 A GRAVITY-INSPIRED MODEL FOR DIRECTED GRAPH AE AND VAE

In this section, we introduce a new model to learn node embeddings from directed graphs using the AE and VAE frameworks, and to address the directed link prediction problem. The main challenge is the following: how to effectively reconstruct asymmetric relationships from encoded representations that are (unique) latent vectors in a node embedding where inner product and standard distances are symmetric?

To overcome this challenge, we resort to classical mechanics and especially to Newton's theory of universal gravitation. We propose an analogy between latent node representations in an embedding and celestial objects in space. Specifically, even if the Earth-Moon distance is symmetric, the *acceleration* of the Moon towards the Earth due to gravity is larger than the acceleration of the Earth towards the Moon. As explained below, this is due to the fact that the Earth is more massive. In the remaining of this section, we transpose

these notions of mass and acceleration to node embeddings to build up our asymmetric graph decoding scheme.

3.1 Newton's Theory of Universal Gravitation

According to Newton's theory of universal gravitation [35], each particle in the universe attracts the other particles through a force called *gravity*. This force is proportional to the product of the masses of the particles, and inversely proportional to the squared distance between their centers. More formally, let us denote by m_1 and m_2 the positive masses of two objects 1 and 2 and by r the distance between their centers. Then, the gravitational force F attracting the two objects is:

$$F = \frac{Gm_1m_2}{r^2},$$

where G is the gravitational constant [7]. Then, using Newton's second law of motion [35], we derive $a_{1\to2}$, the acceleration of object 1 towards object 2 due to gravity:

$$a_{1\to2} = \frac{F}{m_1} = \frac{Gm_2}{r^2}.$$

Likewise, the acceleration $a_{2\to1}$ of 2 towards 1 due to gravity is:

$$a_{2\to1} = \frac{F}{m_2} = \frac{Gm_1}{r^2}.$$

We note that $a_{1\to2} \neq a_{2\to1}$ when $m_1 \neq m_2$. More precisely, we have $a_{1\to2} > a_{2\to1}$ when $m_2 > m_1$ and conversely, i.e. the acceleration of the less massive object towards the more massive object due to gravity is higher.

Despite being superseded in modern physics by Einstein's theory of general relativity [11], describing gravity not as a force but as a consequence of spacetime curvature, Newton's law of universal gravitation is still used in many applications, as the theory provides precise approximations of the effect of gravity when gravitational fields are not extreme. In the following of this paper, we directly draw inspiration from this theory, notably from the formulation of acceleration, to build our proposed autoencoders models. We highlight that Newtonian gravity concepts were already successfully leveraged in [2] for graph visualization, and in [49] where the force formula has been transposed to graph mining measures, to construct symmetric similarity scores among nodes.

3.2 From Physics to Node Representations

Let us come back to our initial analogy between celestial objects in space and node embeddings. In this subsection, let us assume that, in addition to a latent vector z_i of dimension $d \ll n$, we have at our disposal a model that is also able to learn a new *mass parameter* $m_i \in \mathbb{R}^+$ for each node $i \in \mathcal{V}$ of a directed graph. Such parameter would capture the propensity of i to attract other nodes from its neighborhood in this graph, i.e. to make them point towards i through a directed edge. From such augmented model, we could apply Newton's equations in the resulting embedding. Specifically, we could use the *acceleration* $a_{i\to j} = \frac{Gm_j}{r^2}$ of a node i towards a node j due to gravity in the embedding as an indicator of the likelihood that i is connected to j in the directed graph, with $r^2 = \|z_i - z_j\|_2^2$. In a nutshell:

- The numerator captures the fact that some nodes are more influential than others in the graph. For instance, in a scientific publications citation network, seminal groundbreaking articles and more influential and should be more cited. Here, the bigger m_j the more likely i will be connected to j via the (i, j) directed edge.
- The denominator highlights that nodes with structural proximity in the graph, typically with a common neighborhood, are more likely to be connected, provided that the model effectively manages to embed these nodes *close* to each other in the latent space representation. For instance, in a scientific publications citation network, article i will more likely cite article j if it comes from a similar field of study.

More precisely, instead of directly dealing with $a_{i \to j}$, we use $\log a_{i \to j}$ in the remaining of this paper. Taking the logarithm has two advantages. Firstly, thanks to its concavity it limits the potentially large values resulting from acceleration towards very central nodes. Also, $\log a_{i \to j}$ can be negative, which is more convenient to reconstruct an unweighted edge (i.e. in the adjacency matrix A we have $A_{ij} = 1$ or 0) using a sigmoid activation function, as follows:

$$
\begin{aligned}
\hat{A}_{ij} &= \sigma(\log a_{i \to j}) \\
&= \sigma(\underbrace{\log Gm_j}_{\tilde{m}_j} - \log \|z_i - z_j\|_2^2)
\end{aligned}
$$

3.3 Gravity-Inspired Directed Graph AE

For pedagogical purposes, we assumed in subsection 3.2 that we had at our disposal a model able to learn mass parameters m_i for all $i \in \mathcal{V}$. Let us know detail how we actually derive such parameters, using the graph autoencoder framework.

3.3.1 Encoder. For the encoder part of the model, we resort to a Graph Convolutional Network processing A and, potentially, a node features matrix X. Such GCN assigns a vector of size $(d + 1)$ to each node of the graph, instead of d as in standard graph autoencoders. The first d dimensions correspond to the latent vector representation of the node i.e. z_i, where $d \ll n$ is the dimension of the node embedding. The last value of the output vector is the model's estimate of $\tilde{m}_i = \log Gm_i$. To sum up, we have:

$$
(Z, \tilde{M}) = GCN(A, X),
$$

where Z is the $n \times d$ matrix of all latent vectors z_i, $\tilde{M}$ is the n-dimensional vector of all values of $\tilde{m}_i$, and $(Z, \tilde{M})$ is the $n \times (d + 1)$ matrix row-concatenating Z and $\tilde{M}$. We note that learning $\tilde{m}_i$ is equivalent to learning m_i, but is also more convenient since we get rid of the gravitational constant G and of the logarithm.

In this GCN encoder, as we process directed graphs, we replace the usual symmetric normalisation of A, i.e. $D^{-1/2}(A + I)D^{-1/2}$, by the out-degree normalization $D_{\text{out}}^{-1}(A + I)$. Here, D_{out} denotes the diagonal out-degree matrix of $A + I$, i.e. the element (i, i) of D_{out} corresponds to the number of edges (potentially weighted) going out of node i, plus one. Therefore, at each layer of the GCN, the feature vector of a node is the average of features vectors from previous layer of the neighbors to which it points, together with its own feature vector and with a ReLU activation.

3.3.2 Decoder. We leverage the previously defined logarithmic version of acceleration, together with a sigmoid activation, to reconstruct the adjacency matrix A from Z and $\tilde{M}$. Denoting $\hat{A}$ the reconstruction of A, we have:

$$
\hat{A}_{ij} = \sigma(\tilde{m}_j - \log \|z_i - z_j\|_2^2).
$$

Contrary to the inner product decoder, we usually have $\hat{A}_{ij} \neq \hat{A}_{ji}$. This approach is therefore more relevant for directed graph reconstruction. Model training is similar to standard graph AE, i.e. we aim at minimizing the reconstruction loss from matrix A, formulated as a weighted cross entropy loss as in [25], by stochastic gradient descent.

3.4 Gravity-Inspired Directed Graph VAE

We also propose to extend our gravity-inspired method to the graph variational autoencoder framework.

3.4.1 Encoder. We extend [25] to build up an inference model for $(Z, \tilde{M})$. In other words, the $(d + 1)$-dimensional latent vector associated to each node i is $(z_i, \tilde{m}_i)$, concatenating the f-dimensional vector z_i and the scalar $\tilde{m}_i$. We have:

$$
q((Z, \tilde{M})|A, X) = \prod_{i=1}^{n} q((z_i, \tilde{m}_i)|A, X),
$$

with Gaussian hypotheses, as [25]:

$$
q((z_i, \tilde{m}_i)|A, X) = \mathcal{N}((z_i, \tilde{m}_i)|\mu_i, \text{diag}(\sigma_i^2)).
$$

Parameters of Gaussian distributions are learned using two GCNs, with similar out-degree normalization w.r.t. subsection 3.3:

$$
\mu = GCN_\mu(A, X) \quad \text{and} \quad \log \sigma = GCN_\sigma(A, X).
$$

3.4.2 Decoder. From sample vectors $(z_i, \tilde{m}_i)$ from these distributions, we then incorporate our gravity-inspired decoding scheme into the generative model attempting to reconstruct A:

$$
p(A|Z, \tilde{M}) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(A_{ij}|z_i, z_j, \tilde{m}_j)
$$

where:

$$
p(A_{ij} = 1|z_i, z_j, \tilde{m}_j) = \sigma(\tilde{m}_j - \log \|z_i - z_j\|_2^2).
$$

As [25], we train the model by maximizing the ELBO of the model's likelihood using full-batch gradient descent and with a Gaussian prior $p((Z, \tilde{M})) = \prod_i p(z_i, m_i) = \prod_i \mathcal{N}((z_i, m_i)|0, I)$. We discuss these Gaussian assumptions in the experimental part of this paper.

3.5 Generalization of the Decoding Scheme

We point out that one can improve the flexibility of our decoding scheme, both in the AE and VAE settings, by introducing an additional parameter $\lambda \in \mathbb{R}^+$ and reconstruct $\hat{A}_{ij}$ as follows:

$$
\hat{A}_{ij} = \sigma(\tilde{m}_j - \lambda \log \|z_i - z_j\|_2^2).
$$

Decoders from sections 3.3 and 3.4 are special cases where $\lambda = 1$. This parameter can be tuned by cross-validation on link prediction tasks (see section 4). The interpretation of such parameter is twofold. Firstly, it constitutes a simple tool to balance the relative importance of the node distance in the embedding for reconstruction w.r.t. the mass attraction parameter. Then, from a physical point of view, it is

equivalent to replacing the squared distance in Newton's formula by a distance to the power of 2λ. In our experimental analysis on link prediction, we provide insights on when and why deviating from Newton's actual theory (i.e. $\lambda = 1$) is relevant.

3.6 On Complexity and Scalability

Assuming featureless nodes, a sparse representation of adjacency matrix A with m non-zero entries, and considering that our models return a dense $n \times (d + 1)$ embedding matrix Z, then the space complexity of our approach is $O(m + n(d + 1))$, both in the AE and VAE frameworks. If nodes also have features to stack up in the $n \times f$ matrix X, then the space complexity becomes $O(m + n(f + d + 1))$, with $d \ll n$ and $f \ll n$ in practice. Therefore, as standard graph AE and VAE models [25], space complexity increases linearly w.r.t. the size of the graph.

Moreover, due to the pairwise computations of L_2 distances between all d-dimensional vectors z_i and z_j involved in our gravity-inspired decoding scheme, our models have a quadratic time complexity $O(dn^2)$ w.r.t. the number of nodes in the graph, as standard graph AE and VAE. As a consequence we focus on medium-size datasets, i.e. graphs with thousands of nodes and edges, in our experimental analysis. We nevertheless point out that extending our model to very large graphs (with millions of nodes and edges) could be achieved by applying the degeneracy framework proposed in [41] to scale graph autoencoders, or a variant of their approach involving directed graph degeneracy concepts [14]. Future works will provide a deeper investigation of these scalability concerns.

4 EXPERIMENTAL ANALYSIS

In this section, we empirically evaluate and discuss the performance of our models, on three real-world datasets and on three variants of the directed link prediction problem.

4.1 Three Directed Link Prediction Tasks

We consider the following three learning tasks for our experiments.

4.1.1 Task 1: General Directed Link Prediction. The first task is referred to as *general directed link prediction.* As previous works [17, 24, 38, 41], we train models on incomplete versions of graphs where 15% of edges were randomly removed. We take directionality into account in the masking process. In other words, if a link between node i and j is reciprocal, we can possibly remove the (i, j) edge but still observe the reverse (j, i) edge in the training incomplete graph. Then, we create validation and test sets from removed edges and from the same number of randomly sampled pairs of unconnected nodes. We evaluate the performance of our models on a binary classification task consisting in discriminating the actual removed edges from the fake ones, and compare results using the AUC and AP scores. In the following, the validation set contains 5% of edges, and the test set contains 10% of edges. Validation set is only used for hyperparameters tuning.

This setting corresponds to the most general formulation of link prediction. However, due to the large number of unconnected pairs of nodes in most real-world graphs, we expect the impact of directionality on performances to be limited. Indeed, for each actual unidirectional edge (i, j) from the graph, it is unlikely to retrieve the reverse (unconnected) pair (j, i) among negative samples in test

Table 1: Directed graphs used in our experiments

Dataset	Number of nodes	Number of edges	Percentage of reciprocity
Cora	2 708	5 429	2.86%
Citeseer	3 327	4 732	1.20%
Google	15 763	171 206	14.55%

set. As a consequence, models focusing on graph proximity and ignoring the direction of the link, such as standard graph AE and VAE, might still perform fairly on such task.

For this reason, in the following of this subsection we also propose two additional learning tasks, designed to reinforce the importance of directionality learning.

4.1.2 Task 2: Biased Negative Samples (B.N.S.) Link Prediction. For the second task, we also train models on incomplete versions of graphs where 15% of edges were removed: 5% for validation set and 10% for test set. However, removed edges are all *unidirectional*, i.e. (i, j) exists but not (j, i). In this setting, the reverse node pairs are included in validation and test sets and constitute negative samples. In other words, all node pairs from validation and test sets are included in *both* directions. As for *general directed link prediction* task, we evaluate the performance of our models on a binary classification task consisting in discriminating actual edges from fake ones, and therefore evaluate the ability of our models to correctly reconstruct $A_{ij} = 1$ and $A_{ji} = 0$ *simultaneously*.

This task has been presented in [59] under the name *biased negative samples link prediction.* It is more challenging w.r.t. general link direction, as the ability to reconstruct asymmetric relationships is more crucial. Therefore, models ignoring directionality and only learning from symmetric graph proximity, such as standard graph AE and VAE, will fail in such setting.

4.1.3 Task 3: Bidirectionality Prediction. As a third task, we evaluate the ability of our models to discriminate *bidirectional* edges, i.e. reciprocal connections, from *unidirectional* edges. Specifically, we create an incomplete training graph by removing at random one of the two directions of all bidirectional edges. Therefore, the training graph only has unidirectional connections. Then, a binary classification problem is once again designed, aiming at retrieving bidirectional edges in a test set composed of their removed direction and of the same number of reverse directions from unidirectional edges (that are therefore fake edges). In other words, for each pair of nodes i, j from the test set, we observe a connection from j to i in the incomplete training graph, but only half of them are reciprocal. This third evaluation task, referred to as *bidirectionality prediction* in this paper, also strongly relies on directionality learning. As a consequence, as for task 2, standard graph AE and VAE are expected to perform poorly.

4.2 Experimental Setting

4.2.1 Datasets. We provide experiments on three publicly available real-world directed graphs, whose statistics are presented in Table 1. The *Cora*[2] and *Citeseer*[2] datasets are *citation graphs* consisting of

---

[2]https://linqs.soe.ucsc.edu/data

Table 2: Directed Link Prediction on Cora, Citeseer and Google graphs

Dataset	Model	Task 1: General Link Prediction		Task 2: B.N.S. Link Prediction		Task 3: Bidirectionality Prediction	
		AUC (in %)	AP (in %)	AUC (in %)	AP (in %)	AUC (in %)	AP (in %)
Cora	*Gravity Graph VAE (ours)*	91.92 ± 0.75	**92.46 ± 0.64**	**83.33 ± 1.11**	**84.50 ± 1.24**	75.00 ± 2.10	73.87 ± 2.82
	Gravity Graph AE (ours)	87.79 ± 1.07	90.78 ± 0.82	**83.18 ± 1.12**	**84.09 ± 1.16**	**75.57 ± 1.90**	73.40 ± 2.53
	Standard Graph VAE	82.79 ± 1.20	86.69 ± 1.08	50.00 ± 0.00	50.00 ± 0.00	58.12 ± 2.62	59.70 ± 2.08
	Standard Graph AE	81.34 ± 1.47	82.10 ± 1.46	50.00 ± 0.00	50.00 ± 0.00	53.07 ± 3.09	54.60 ± 3.13
	Source/Target Graph VAE	85.34 ± 1.29	88.35 ± 0.99	63.00 ± 1.05	64.62 ± 1.37	**75.20 ± 2.62**	**73.86 ± 3.04**
	Source/Target Graph AE	82.67 ± 1.42	83.25 ± 1.51	57.81 ± 2.64	57.66 ± 3.35	65.83 ± 3.87	63.15 ± 4.58
	APP	**93.92 ± 1.01**	**93.26 ± 0.60**	69.20 ± 0.65	67.93 ± 1.09	72.85 ± 1.91	70.97 ± 2.60
	HOPE	80.82 ± 1.63	81.61 ± 1.08	61.84 ± 1.84	63.73 ± 1.12	65.11 ± 1.40	64.24 ± 1.18
	node2vec	79.01 ± 2.00	84.20 ± 1.62	50.00 ± 0.00	50.00 ± 0.00	66.97 ± 1.41	67.61 ± 1.80
Citeseer	*Gravity Graph VAE (ours)*	87.67 ± 1.07	89.79 ± 1.01	**76.19 ± 1.35**	**79.27 ± 1.24**	**71.61 ± 3.20**	**71.87 ± 3.87**
	Gravity Graph AE (ours)	78.36 ± 1.55	84.75 ± 1.10	**75.32 ± 1.53**	**78.47 ± 1.27**	**71.48 ± 3.64**	**71.50 ± 3.62**
	Standard Graph VAE	78.56 ± 1.43	83.66 ± 1.09	50.00 ± 0.00	50.00 ± 0.00	47.66 ± 3.73	50.31 ± 3.27
	Standard Graph AE	75.23 ± 2.13	75.16 ± 2.04	50.00 ± 0.00	50.00 ± 0.00	45.01 ± 3.75	49.79 ± 3.71
	Source/Target Graph VAE	79.45 ± 1.75	83.66 ± 1.32	57.32 ± 0.92	61.02 ± 1.37	69.67 ± 3.12	67.05 ± 4.10
	Source/Target Graph AE	73.97 ± 3.11	75.03 ± 3.37	56.97 ± 1.33	57.62 ± 2.62	54.88 ± 6.02	55.81 ± 4.93
	APP	**88.70 ± 0.92**	**90.29 ± 0.71**	64.35 ± 0.45	63.70 ± 0.51	64.16 ± 1.90	63.77 ± 3.28
	HOPE	72.91 ± 0.59	71.29 ± 0.52	60.24 ± 0.51	61.28 ± 0.57	52.65 ± 3.05	54.87 ± 1.67
	node2vec	71.02 ± 1.78	77.70 ± 1.22	50.00 ± 0.00	50.00 ± 0.00	61.08 ± 1.88	63.63 ± 2.77
Google	*Gravity Graph VAE (ours)*	**97.84 ± 0.25**	98.18 ± 0.14	**88.03 ± 0.25**	**91.04 ± 0.14**	84.69 ± 0.31	84.89 ± 0.30
	Gravity Graph AE (ours)	97.77 ± 0.10	**98.43 ± 0.10**	**87.71 ± 0.29**	**90.84 ± 0.16**	**85.82 ± 0.63**	**85.91 ± 0.50**
	Standard Graph VAE	87.14 ± 1.20	88.14 ± 0.98	50.00 ± 0.00	50.00 ± 0.00	40.03 ± 4.98	44.69 ± 3.52
	Standard Graph AE	91.34 ± 1.13	92.61 ± 1.14	50.00 ± 0.00	50.00 ± 0.00	41.35 ± 1.92	41.92 ± 0.81
	Source/Target Graph VAE	96.33 ± 1.04	96.24 ± 1.06	85.30 ± 3.18	84.69 ± 4.42	75.11 ± 2.07	73.63 ± 2.06
	Source/Target Graph AE	**97.76 ± 0.41**	**97.74 ± 0.40**	86.16 ± 2.95	86.26 ± 3.33	82.27 ± 1.29	80.10 ± 1.80
	APP	97.04 ± 0.10	96.97 ± 0.11	83.06 ± 0.46	85.15 ± 0.42	73.43 ± 0.16	68.74 ± 0.19
	HOPE	81.16 ± 0.67	83.02 ± 0.35	74.23 ± 0.80	72.70 ± 0.79	70.45 ± 0.18	70.84 ± 0.22
	node2vec	83.11 ± 0.27	85.79 ± 0.30	50.00 ± 0.00	50.00 ± 0.00	78.99 ± 0.35	76.72 ± 0.53

scientific publications citing one another. The *Google*[3] dataset is a *web graph*, whose nodes are web pages and directed edges represent hyperlinks between them. The *Google* graph is denser than *Cora* and *Citeseer* and has a higher proportion of bidirectional edges. Graphs are unweighted and featureless.

4.2.2 Standard and Gravity-Inspired Autoencoders. We train gravity-inspired AE and VAE models for each graph. For comparison purposes, we also train standard graph AE and VAE from [25]. Each of these four models includes a two-layer GCN encoder with 64-dim hidden layer and with out-degree left normalization of A as defined in subsection 3.3.1. All models are trained for 200 epochs and return 32-dim latent vector node representations. We use Adam optimizer [22], apply a learning rate of 0.1 for *Cora* and *Citeseer* and 0.2 for *Google*, train models without dropout, performing full-batch gradient descent and using the reparameterization trick [23] for variational autoencoders. Also, for *tasks 1 and 3* we picked $\lambda = 1$ (respectively $\lambda = 10$) for *Cora* and *Citeseer* (resp. for *Google*) ; for *task 2* we picked $\lambda = 0.05$ for all three graphs, which we interpret in next subsections. All hyperparameters were tuned from AUC score on *task 1* i.e. on general directed link prediction task.

4.2.3 Baselines. Besides standard AE and VAE models, we also compare the performance of our methods w.r.t. the alternative graph embedding methods introduced in subsection 2.6:

- Our *Source/Target Graph AE and VAE*, extending the source target vectors paradigm to graph AE and VAE, and trained with similar settings w.r.t. standard and gravity models.

[3]http://konect.uni-koblenz.de/networks/cfinder-google

- HOPE [36], setting $\beta = 0.01$ and with source and target vectors of dimension 16, to learn 32-dim node representations.
- APP [59], training models over 100 iterations to learn 16-dim source and target vectors, i.e. 32-dim node representations, with standard settings from [59]'s implementation.
- For comparison purposes, in our experiments we also train *node2vec* models [16] that, while dealing with directionality in random walks, only return one 32-dim embedding vector per node. We rely on *symmetric* inner products with sigmoid activation for link prediction, and we therefore expect node2vec to underperform w.r.t. APP. We trained models from 10 random walks of length 80 per node, with $p = q = 1$ and a window size of 5.

We used Python and especially the Tensorflow library, except for APP where we used the authors' Java implementation [59]. We trained models on a NVIDIA GTX 1080 GPU and ran other operations on a double Intel Xeon Gold 6134 CPU.

4.3 Results for Directed Link Prediction

Table 2 reports mean AUC and AP, along with standard errors over 100 runs, for each dataset and for the three tasks. Train incomplete graphs and test sets are different for each of the 100 runs. Overall, our gravity-inspired graph AE and VAE models achieve very competitive results.

On *task 1*, standard graph AE and VAE, despite ignoring directionality for graph reconstruction, still perform fairly well (e.g. 82.79% AUC for standard graph VAE on Cora). This emphasizes the limited impact of directionality on performances for such task, as
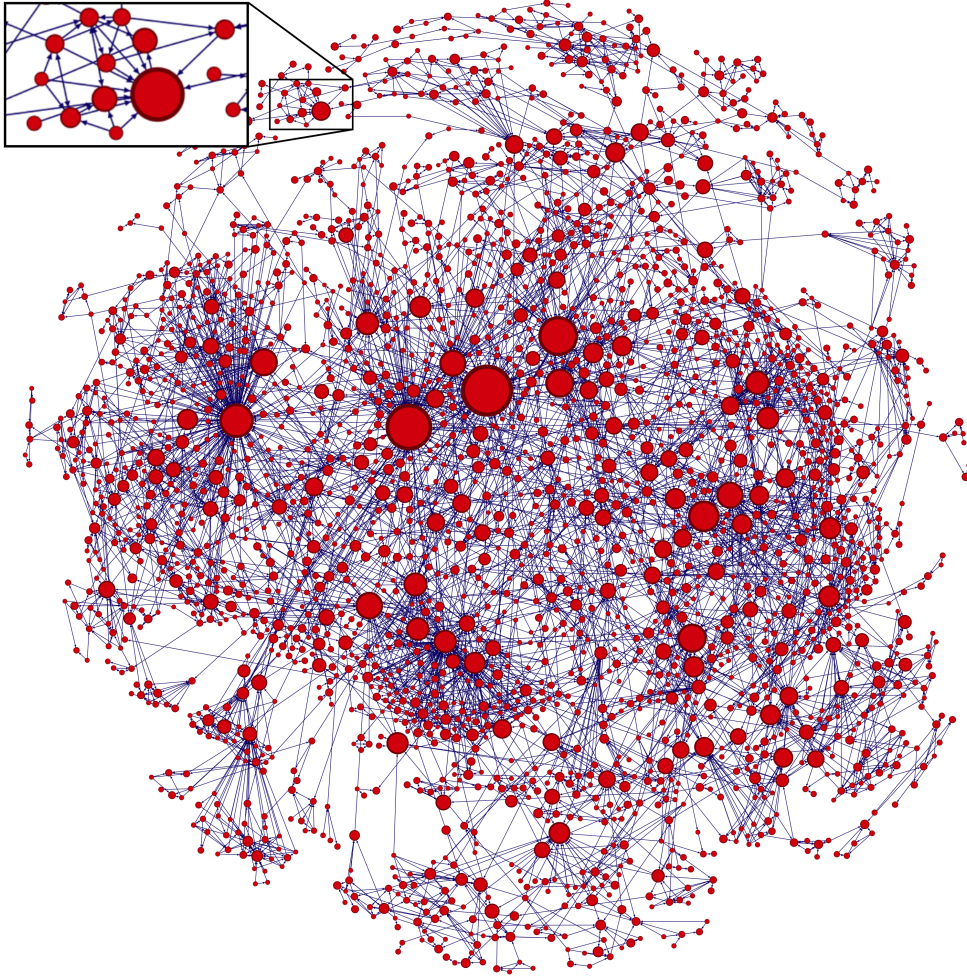
Figure 1: Visualization of Cora graph based on embeddings learned from gravity-inspired graph VAE model. In this graph, nodes are scaled using mass parameter $\tilde{m}_i$. Node separation is based on distances in the embedding, using Force Atlas 2 layout and Fruchterman-Reingold algorithm [12] on Gephi. Edges directionalities are best viewed on screen.

planned in subsection 4.1.1. Nonetheless, our gravity-inspired models significantly outperform the standard ones (e.g. 91.92% AUC for gravity-inspired graph VAE on Cora), confirming the relevance of capturing both proximity and directionality for general directed link prediction. Moreover, our models are competitive w.r.t. baselines designed for directed graphs. Among them, APP is the best on our three datasets, together with the source/target graph AE on Google graph.

On *task 2*, i.e. biased negative samples link prediction, our gravity-inspired models persistently achieve the best performances (e.g. a top 76.19% AUC on Citeseer, 11+ points above the best baseline). We notice that models ignoring directionality for prediction, i.e. node2vec and standard graph AE and VAE, totally fail (50.00% AUC and AP on all graphs, corresponding to the random classifier level) which was expected since test sets include both directions of each node pair. Experiments on *task 3*, i.e. on bidirectionality prediction, confirm the superiority of our approach when dealing with challenging tasks where directionality learning is crucial. Indeed, on

this last task, gravity-inspired models also outperform alternative approaches (e.g. with a top 85.82% AUC for gravity-inspired graph AE on Google).

While the AE and VAE frameworks are based on different foundations, we found no significant performance gap in our experiments between (standard, asymmetric, or gravity-inspired) autoencoders and their variational counterparts. This result is consistent with previous insights from [25, 41] on undirected graphs. Future works will investigate alternative prior distributions for graph VAE, aiming at challenging the traditional Gaussian hypothesis that, despite being very convenient for computations, might not be an optimal choice in practice [25]. Last, we note that all AE/VAE models required a comparable training time of roughly 7 seconds (respectively 8 seconds, 5 minutes) for Cora (resp. for Citeseer, for Google) on our machine. Baselines were faster: for instance, on the largest Google graph, 1 minute (resp. 1.30 minutes, 2 minutes) were required to train HOPE (resp. APP, node2nec).
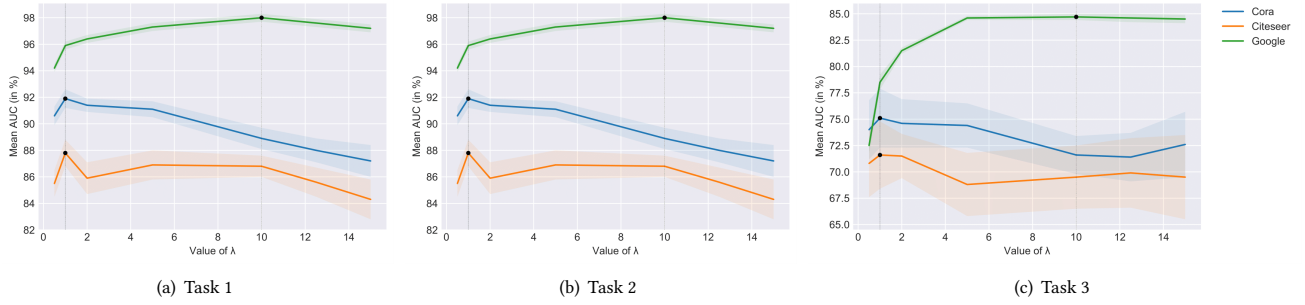
| (a) Task 1 | (b) Task 2 | (c) Task 3 |

Figure 2: Impact of parameter λ on mean AUC, ±1 standard error, for gravity-inspired graph VAE

Table 3: Pearson correlation coefficient of centrality measures with parameter $\tilde{m}_i$, learned from gravity-inspired graph VAE - Katz on Google not reported due to complexity.

Centrality Measures	Cora	Citeseer	Google
In-degree	0.5960	0.6557	0.1571
Out-degree	-0.2662	-0.1994	0.0559
Betweenness	0.5370	0.4945	0.2223
Pagerank	0.4143	0.3715	0.1831
Katz	0.5886	0.6428	-

4.4 Discussion

To pursue our experimental analysis, we propose a discussion on the nature of $\tilde{m}_i$, on the role of λ to balance node proximity and influence, and on some limits and openings of our work.

4.4.1 Deeper insights on $\tilde{m}_i$. Figure 1 displays a visualization of Cora graph, using embeddings and $\tilde{m}_i$ parameters learned through our gravity-inspired graph VAE. In such visualization, we observe that nodes with smaller "masses" tend to be connected to nodes with larger "masses" from their embedding neighborhood, which was expected by design of our decoding scheme.

From Figure 1, one might argue that $\tilde{m}_i$ tend to reflect centrality. In this direction, we compare $\tilde{m}_i$ to the most common graph centrality measures. Specifically, in Table 3 we report Pearson correlation coefficients of $\tilde{m}_i$ w.r.t. the following measures:

- The *in-degree* and *out-degree* of the node, that are respectively the number of edges coming into and going out of the node.
- The *betweenness centrality*, which is, for a node i, the sum of the fraction of all pairs shortest paths that goes through $i : c_B(i) = \sum_{s,t \in \mathcal{V}} \frac{sp(s,t|i)}{sp(s,t)}$, where $sp(s,t)$ is the number of shortest paths from node s to node t, and $sp(s,t|i)$ is the number of those paths going through i [5].
- The *PageRank* [37], computing a ranking of nodes importances based on the structure of incoming links. It was originally designed to rank web pages, and it can be seen as the stationary distribution of a random walk on the graph.
- The *Katz centrality*, a generalization of the eigenvector centrality. Katz centrality of node i is $c_i = \alpha \sum_{1 \le j \le n} A_{ij} c_j + \beta$, where A is the adjacency matrix with largest eigenvalue $\lambda_{\max}$, with usually $\beta = 1$ and with $\alpha < \frac{1}{\lambda_{\max}}$ [20].

As observed in Table 3, the parameter $\tilde{m}_i$ is positively correlated to all of these centrality measures, except for out-degree where the correlation is negative (or almost null for Google), meaning that nodes with few edges going out of them tend to have larger values of $\tilde{m}_i$. Correlations are not perfect which emphasizes that our models do not exactly learn one of these measures. We also note that centralities are lower for Google, which might be due to the structure of this graph and especially to its density.

In our experiments, we tried to replace $\tilde{m}_i$ by any of these (normalized) centrality measures when performing link prediction, and to learn optimal vectors z_i for these fixed masses values, achieving underperforming results. For instance, we reached a 89.05% AUC by using betweenness centrality on Cora instead of the actual $\tilde{m}_i$ learned by the VAE, which is above standard graph VAE (82.79% AUC) but below the gravity VAE with $\tilde{m}_i$ (91.92% AUC). Also, using centrality measures as initial values for $\tilde{m}_i$ before model training did not significantly impact performances in experiments.

4.4.2 Impact of parameter λ. In subsection 3.5 we introduced a parameter $\lambda \in \mathbb{R}^+$ to tune the relative importance of node proximity w.r.t. mass attraction, leading to the reconstruction scheme $\hat{A}_{ij} = \sigma(\tilde{m}_j - \lambda \log \|z_i - z_j\|_2^2)$. In Figure 2, we show the impact of λ on mean AUC scores for the VAE model and for all three datasets. For Cora and Citeseer, on *task 1* and *task 3*, $\lambda = 1$ is an optimal choice, consistently with Newton's formula (see Figure 2 (a) and (c)). However, for Google, on *task 1* and *task 3*, we obtained better performances for higher values of λ, notably for $\lambda = 10$ that we used in our experiments. Increasing λ reinforces the relative importance of node symmetric proximity in the decoder, measured by $\log \|z_i - z_j\|_2^2$, w.r.t. parameter $\tilde{m}_j$ capturing the global influence of a node on its neighbors and therefore asymmetries in links. Since the Google graph is much denser than Cora and Citesser, and has a higher proportion of symmetric relationships (see Table 1), putting the emphasis on node proximity appears as a relevant strategy.

On a contrary, on *task 2* we achieved optimal performances for $\lambda = 0.05$, for all three graphs (see Figure 2 (b)). Since $\lambda < 1$, we therefore improved scores by assigning more relative importance to the mass parameter $\tilde{m}_j$. Such result is not surprising since, for *biased negative samples link prediction* task, learning directionality is more crucial than learning proximity, as nodes pairs from test sets are all included in both directions. As display in Figure 2 (b), increasing λ significantly deteriorates performances.

4.4.3 Extensions and openings. Throughout these experiences, we focused on featureless graphs, to fairly compete with HOPE, APP and node2vec. However, as explained in section 3, our models can easily leverage node features, in addition to the graph structure summarized in *A*. Moreover, the gravity-inspired method is not limited to GCN encoder and can be generalized to any alternative graph neural network. Future works will provide more evidence on such extensions, will investigate better-suited priors for graph VAE, and will generalize existing scalable graph AE/VAE framework [41] to directed graphs. We also aim at exploring to which extent graph AE/VAE can tackle the node clustering problem in directed graphs.

5 CONCLUSION

In this paper we presented a new method, inspired from Newtonian gravity, to learn node embeddings from directed graphs, using graph AE and VAE. We provided experimental evidences of its ability to effectively address the challenging directed link prediction problem. Our work also pinpointed several research directions that, in the future, should lead towards the improvement of our approach.

REFERENCES

[1] Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. *ICML workshop on unsupervised and transfer learning.*
[2] Michael J Bannister, David Eppstein, Michael T Goodrich, and Lowell Trott. 2012. Force-directed graph drawing using social gravity and scaling. In *International Symposium on Graph Drawing.*
[3] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15, 6, 1373–1396.
[4] Rianne van den Berg, Thomas N. Kipf, and Max Welling. 2018. Graph convolutional matrix completion. *KDD Deep Learning day.*
[5] Ulrik Brandes. 2008. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks* 30, 2, 136–145.
[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. *ICLR.*
[7] Henry Cavendish. 1798. XXI. Experiments to determine the density of the earth. *Philosophical Transactions of the Royal Society of London* 88, 469–526.
[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *ICLR.*
[9] Michael Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *NIPS.*
[10] Tien Huu Do, Duc Minh Nguyen, Evaggelia Tsiligianni, Angel Lopez Aguirre, Valerio Panzica La Manna, Frank Pasveer, Wilfried Philips, and Nikos Deligiannis. 2019. Matrix Completion with Variational Graph Autoencoders: Application in Hyperlocal Air Quality Inference. *ICASSP.*
[11] Albert Einstein. 1915. Erklarung der Perihelionbewegung der Merkur aus der allgemeinen Relativitatstheorie. *Sitzungsber. preuss. Akad. Wiss., vol. 47, No. 2, pp. 831-839, 1915* 47, 831–839.
[12] Thomas MJ Fruchterman and Edward M Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and experience* 21, 11, 1129–1164.
[13] Dario Garcia Gasulla. 2015. Link prediction in large directed graphs. Ph.D. thesis, Universitat Politècnica de Catalunya.
[14] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2013. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems* 35, 2, 311–343.
[15] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep learning.* MIT press.
[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. *SIGKDD.*
[17] Aditya Grover, Aaron Zweig, and Stefano Ermon. 2018. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459.*
[18] Sogol Haghani and Mohammad Reza Keyvanpour. 2017. A systemic analysis of link prediction in social network. *Artificial Intelligence Review,* 1–35.
[19] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. *ICML.*
[20] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1, 39–43.
[21] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 137–146.
[22] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR.*

[23] D. P. Kingma and M. Welling. 2013. Auto-encoding variational bayes. *ICLR.*
[24] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *ICLR.*
[25] Thomas N. Kipf and Max Welling. 2016. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning.*
[26] I. A. Kovács, K. Luck, K. Spirohn, Y. Wang, C. Pollis, S. Schlabach, W. Bian, D.-K. Kim, N. Kishore, T. Hao, et al. 2019. Network-based prediction of protein interactions. *Nature communications* 10, 1, 1240.
[27] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1, 79–86.
[28] Jing Li, Lingling Zhang, Fan Meng, and Fenhua Li. 2014. Recommendation algorithm based on link prediction and domain knowledge in retail transactions. *Procedia Computer Science* 31, 875–881.
[29] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7, 1019–1031.
[30] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. 2018. Constrained Graph Variational Autoencoders for Molecule Design. *NeurIPS.*
[31] Tengfei Ma, Jie Chen, and Cao Xiao. 2018. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. *NeurIPS.*
[32] F. Malliaros, C. Giatsidis, A. Papadopoulos, and M. Vazirgiannis. 2019. The Core Decomposition of Networks: Theory, Algorithms and Applications. *hal-01986309.*
[33] Fragkiskos D Malliaros and Michalis Vazirgiannis. 2013. Clustering and community detection in directed networks: A survey. *Physics Reports* 533, 4, 95–142.
[34] Kurt Miller, Michael I Jordan, and Thomas L Griffiths. 2009. Nonparametric latent feature models for link prediction. In *NIPS.*
[35] Isaac Newton. 1687. *Philosophiae naturalis principia mathematica.*
[36] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. *SIGKDD.*
[37] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: Bringing order to the web. *Stanford InfoLab.*
[38] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially Regularized Graph Autoencoder. *IJCAI.*
[39] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. *SIGKDD.*
[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. *Parallel Distributed Processing, Vol 1.*
[41] Guillaume Salha, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. 2019. A Degeneracy Framework for Scalable Graph Autoencoders. *IJCAI.*
[42] B. Samanta, A. De, G. Jana, P. K. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez. 2018. NeVAE: A Deep Generative Model for Molecular Graphs. *AAAI.*
[43] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The graph neural network model. *Neural Networks* 20, 1, 61–80.
[44] Daniel Schall. 2015. Link prediction for directed graphs. In *Social Network-Based Recommender Systems.* Springer, 7–31.
[45] Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *ICANN.*
[46] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. *WWW.*
[47] Phi Vu Tran. 2018. Learning to Make Predictions on Graphs with Autoencoders. *DSAA.*
[48] M. Tschannen, O. Bachem, and M. Lucic. 2018. Recent Advances in Autoencoder-Based Representation Learning. *NeurIPS Bayesian Deep Learning workshop.*
[49] Akanda Wahid-Ul-Ashraf, Marcin Budka, and Katarzyna Musial-Gabrys. 2017. NewtonâĂŹs gravitational law for link prediction in social networks. In *International Conference on Complex Networks and their Applications.*
[50] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. *CIKM.*
[51] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. *SIGKDD.*
[52] P. Wang, B. Xu, Y. Wu, and X. Zhou. 2015. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences* 58, 1, 1–38.
[53] F. Wu, T. Zhang, A. H. de Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. *ICML.*
[54] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596.*
[55] Yan Yu and Xinxin Wang. 2014. Link prediction in directed network and its application in microblog. *Mathematical Problems in Engineering.*
[56] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. Network representation learning: A survey. *IEEE transactions on Big Data.*
[57] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen. 2019. D-VAE: A Variational Autoencoder for Directed Acyclic Graphs. *arXiv preprint arXiv:1904.11088.*
[58] D. Zhao, L. Zhang, and W. Zhao. 2016. Genre-based link prediction in bipartite graph for music recommendation. *Procedia Computer Science* 91, 959–965.
[59] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. In *AAAI.*