Graph Embeddings for Social Network Analysis: State of the Art

Paul Compagnon, Kilian Ollivier
INSA Lyon - Département Informatique
Universität Passau

Supervised by:
Lionel Brunie
Didier Puzenat
Konstantin Ziegler

Contents

Chapter I

Introduction

Social network analysis (SNA) started as a subject of social sciences. The goal is to describe social interactions inside social structures (*e.g.* associations, companies, universities, *etc*). As a part of social sciences, social network analysis is quite different from traditional tools used in this field : it does not put people into categories (women, men, adults, children, *etc.*) but deals with interactions between people. This idea is summed up by Degenne and Forsé [1]: "Instead of thinking reality in terms of relations between actors, lots of those who analyze empirical data limit themselves into thinking it in categories (for example: young people, women, executives, developing countries, *etc.*). These categories are built by aggregating people with similar features and, *a priori*, relevant for the current issue."

First foundations of social network analysis were built during the forties and the sixties with studies lead by Gluckman about urbanization in Zambia [1]. In 1967, Stanley Milgram conducted his famous social experiment [2, 3], involving letters through USA, and concluded that the distance between two people chosen at random in the human social graph is shorter than we could imagine ("six degrees of separation"). It appears that many of the social networks we can observe in everyday life exhibits this characteristic. It is called a "small-word" network.

Theoretical framework lies upon this principle: individuals are influenced by the social structures[1] they live in [1]. Concerning the methodology, social networks analysis relies on data gathering (both quantitative and qualitative data), data analytics and experiments on models such as small-world graphs.

I.1 Social networks analysis and computer science

Definition 1 *A graph G is composed of a set of vertices V and a set of edges E , formed by pairs of vertices. $V = \{v_1, v_2, v_3, ..., v_n\}, E = \{(v_1, v_2), (v_1, v_3), ...\}$. If the graph is directed, the pairs of vertices that form edges are ordered. A weighted graph is noted $V = (G, E, W)$ where W associates a real value for each edge $W : E \mapsto \mathbb{R}$.*

Social structures and especially social networks can be modeled by a graph. A graph is a mathematical object which is conposed of vertices and edges. Vertices represent actors and edges map how they interact each other. The graph can be directed when interactions are asymmetric (figure I.1) . The interest of computer science for SNA has grown since the rise of popular social websites such as Facebook, Twitter and Linkedin. They brought new challenges and provide large data sets for experiments (the Twitter data collected by the Arizona State University is composed of 11,316,811 nodes and 85,331,846 edges).
As listed by Lei Tang and Huan Liu [4], main tasks of SNA are:

Network modeling aims to simulate the behavior of the network with a simple model. For example, Barabási *et al* made an attempt to model large networks such as the WWW [5] with a simple algorithm.

[1] *social structures* is here taken in a very broad sense

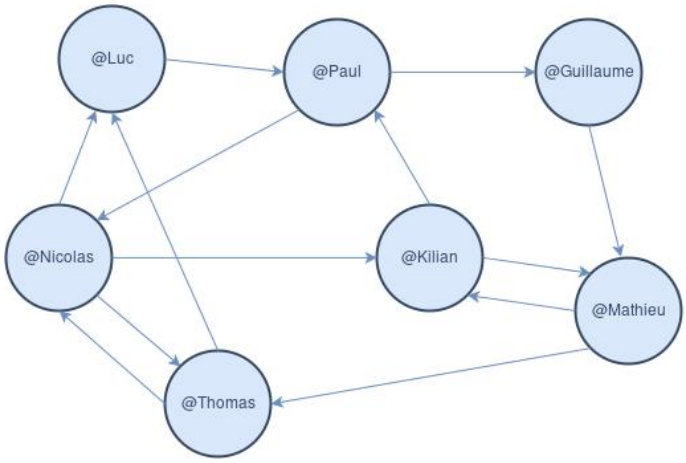


Figure I.1: A simple directed social graph that could belong to Twitter, Instagram, Weibo or any social network with a "following" mechanism.

Information diffusion studying aims to understand how an information is diffused through the network. Social networks owns some properties – for example the node connectivity is not uniform through the network – that makes the diffusion very heterogeneous. Information diffusion is a whole research field, and it can help to solve many various issues such as preventing terrorist attacks and optimizing social marketing campaigns [6][2].

Node classification aims to retrieved individual data of an actor (age, gender, interests *etc.*) [7]. Since two connected actors are likely to share many properties (homophily), we can predict one's missing properties from its neighborhood. For example, Facebook predicts people's education pathway then encourages them to confirm these personal informations.

Link prediction aims to predict the connections that will appear between the actors of a social network. Since social networks evolve quickly and continuously, link prediction is very useful to know what could happen in the near future. Link prediction can help recovering a whole social graph when we only have an incomplete view of it. This is also used for friend recommendation or even viral marketing [8].

Community detection aims to identify groups of nodes highly interconnected. For example it can be a group of high school students, or a group of scientists in a co-authorship network. Identifying network communities allows one to better understand network structure, to discover functionally related objects, and to study interactions between communities [9]. It can also be a first step before studying link prediction and network classification.

Node role identification aims to infer from a social graph the role the node is playing into the network. For example, a node can be a hub (if it has many connections) or a bridge (if it connects two communities). Some roles can be characteristic of a social network, such as the *answer person* in Reddit [10]. Role analysis allow one to identify actors with interesting properties.

Social Network Analysis can be very tough when working on large social networks, because many measures require expensive graph search. For example, identifying the community structure of a network is a very computationally expensive task as stated by this review of 2005 [11]. According to Ou *et al.* [12], "to process graph data effectively, the first critical challenge is graph data representation, that is, how to represent graphs properly so that advanced analytic tasks, such as pattern discovery, analysis, and prediction, can be conducted efficiently in both time and space". Graph embedding is a way of representing graphs with vectors, such that later analysis are made easier. The problem of graph embedding is to preserve the graph inner properties (in our case, the properties used by SNA).

[2]This paper from Guille *et al.* is a fairly complete overview of information diffusion in social networks.

I.2 Graph embedding

Definition 2 *Let $G = (V, E)$ be a graph, where V is the set of vertices and E is the set of edges. Graph embedding is defining an application $\phi : V \mapsto \mathbb{R}^N$. ϕ associates to each node of V a vector representation of length N that preserves some graph properties (e.g. neighborhoods, communities etc.).*

Definition 3 *Let M be the adjacency matrix of a graph G. $M_{i,j} = 1$ if v_i and v_j if $(v_i, v_j) \in E$, otherwise $M_{i,j} = 0$*

In this document, we tackle the issue of graph embedding applied to social networks analysis. Producing vector representations of nodes builds a bridge between social network analysis, data analytics and statistics. The vectors can therefore be used by machine learning algorithms that takes vectors as input for prediction, distance computation or clustering. For example, node classification can be performed using node vectors and classic approaches from machine learning such as logistic regression [13] and Support Vector Machine [14]. Classical tools can also be used to evaluate the classification such as Receiver Operating Characteristic (ROC) curve which gives a good overview of the qualities of the studied classifier [15] when it is compared with the curve produced by a random classifier. Two-dimensional embedding can also be used for visualization, chip design [16] or just to project the graph in a low vector space.

The simplest graph embedding we can consider is the adjacency matrix of the network. As stated before, Social Network Analysis often deals with millions of nodes, therefore the size of the adjacency matrix is the square of number of nodes. Not mentioning obvious performance issues, this embedding would be hardly exploitable because of the curse of dimensionality [3]. More complex embedding approaches have been developed in the context of dimensional reduction. We can list: spectral embedding [20], Laplacian Eigenmap [21], LLE [22]. Studying large network implies a lot of performance issues. Criterias for algorithms to be tractable, are a low complexity (linear or polynomial) or being easily parallelizable. However, these approaches rely on algorithms for matrix eigen decomposition and they are all computationally expensive[4]. Finally, these algorithms have not been designed to produce vector representations of actors in a social network and therefore do not preserve the social structure of the network. Recent approaches based on artificial neural networks came up with good performances. Initially, the neural networks models that are used have been designed for word embedding purpose [24].

In the first chapter, we will explain how neural networks can improve node embedding, and how we can use tools created for word embedding. In the the second chapter, we will explore the three main social networks embedding algorithms, they rely on tools introduced in the first chapters. Then we will tackle the issues of the experimentations and evaluations in the third chapter. In the last chapter, we discuss ideas for future works.

[3] The term refers to an issue appearing with very high dimensional spaces, defined by Bellman in 1957 [17, 18]. The curse of dimensionality can highly compromise statistical analysis. In machine learning tasks, Hughes [19] showed that with a fixed number of training samples, the predictive power decreases as the number of dimensions goes up.

[4] This Wikipedia article [23] references several algorithms for eigen decomposition and their complexities are, at best, quadratic. That's not sufficient enough when working with networks of millions of nodes. Some converge linearly but they produce only approximations of the eigenvalues

Chapter II

From word embedding to graph embedding

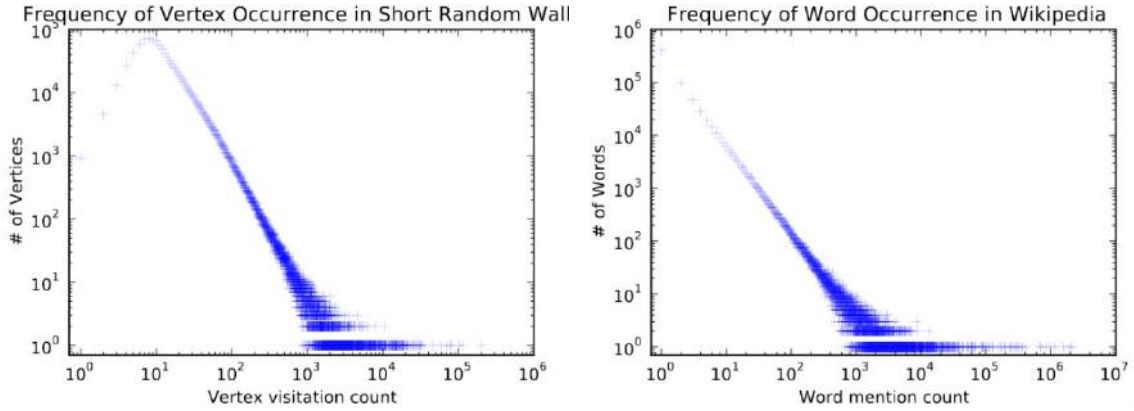
The main difficulty encountered by graph embedding algorithms is preserving the structure of the network while being scalable ; on these issues, previous works on dimensional reduction are not satisfying enough. Recent advances on feature extraction and representational learning [25], especially on word embedding, exhibited tools that can overcome these issues. We will first show why we can build analogies between natural language processing (NLP) and social network analysis. Then, we will introduce Word2Vec [26, 24], a word embedding algorithm, which brought two efficient models based on neural networks: CBOW (Continuous Bag-Of-Words) and Skip-Gram. Some part of these models are hard to compute and this is a challenge for both word embedding and graph embedding. That is why we dedicate a subsection (II.3) to the methods used to bypass these problems. We finally describe the advantages of these models.

II.1 Similarities between Natural Language Processing and Social Network Analysis

Natural language processing deals with human languages and their understanding by computers [27]. Thanks to artificial neural networks, this field has made great advances during the last decades. One of the last major contribution is Word2Vec [24], a word embedding algorithms which exhibits good results and properties. Analogies can be made between word embedding and graph embedding :

	Natural Language Processing	Social Network Analysis
Atomic unit	word	actor (node of the social graph)
Goal	Discover what are the roles of words in a sentence and their relations (both syntactic and semantic).	Understand the role of each actor in the network and how they interact each other.
Main difficulty	Large number of words in a language (up to a hundred thousand)	Large number of actors (up to one billion for Facebook)

If we generate sequences of nodes with short random walks in a social graph, the frequency distribution of nodes will be similar to the word frequency in a large corpus [28]. Both of these distributions follow the power law (Zipf's law). We can understand this distribution for words : a few number of them are overused such as *the*, *of*, *one*, whereas the majority of words are really infrequent. For social networks, the scale-free property is the source of that distribution.



(a) Youtube Social Graph (b) Wikipedia Article Text

Figure II.1: The distributions of vertex occurrence (a) and word occurrence (b) are quite similar, they both follow a power law distribution [28].

Social networks are scale-free

Considering social networks as self-organizing systems, they have been first described with the random graph model [29]. This model builds a graph from N isolated vertices, then adds randomly edges between them with a probability p. However, further studies, for example the mapping of the World Wide Web in 1999 [30] showed that social networks often contains highly connected vertices (called *hubs*). This behavior is not predicted by the random graph model. Barabási and Albert argue that the model must include "growth and preferential attachment" [5]. It means that a real social network takes time to grow and that creation of a new edge depends on the graph structure. If a new actor appears in the network, the connection probability is higher with the hubs than with poorly connected actors. This phenomenon has social explanations. For example, a hub is more likely to be well-known, hence it should drain more connections (it creates a *rich get richer* effect). That is why the connectivity distribution follows the power law, as we can see in figure II.1. Such distributions are independent from the scale (scale-free), meaning that for a distribution $P(x) \sim x^{-\alpha}$ we have $P(ax) = bP(x)$ [4]. Therefore, graphs which obey to this law are also called scale-free. This similarity between social networks and text corpus brought the idea of using natural language processing tools for graph processing (with nodes sequences instead of sentences).

II.2 Word2Vec

Notations We define the vocabulary V as set of words, and $(w_1, w_2, ..., w_{n-1}, w_n) \in V^N$ as an ordered sequence of words . The context of a word is noted $C : V \mapsto V^N$.

II.2.1 Introduction to word embedding

Definition 4 *Let $S = (w_1, w_2, ..., w_n)$ be a sequence of words (a sentence). Word embedding is defining an application $\phi : S \mapsto \mathbb{R}^N$ that associates a real vector representation of length N to a word of S.*

The embedding is designed to preserves some properties of S (for example, two adjacent words must be close in the vector space). Two different word embedding algorithm can preserve different properties, that are captured by the word's context : we analyze a word's relations by looking at its context. The context is a flexible notion whose definition can differ whether what we want to model. It can be the whole sentence or the whole paragraph the word belongs to. The main idea is that similar words tends to appear in similar contexts [31]. If the contexts of two words w_1 and w_2 share many words, but a word w_3 is not in both context, the learning method must ensure that:

$$\begin{cases} dist(\phi(w_1), \phi(w_2)) < dist(\phi(w_1), \phi(w_3)) \\ dist(\phi(w_1), \phi(w_2)) < dist(\phi(w_2), \phi(w_3)) \end{cases}$$

Definition 5 *Let M be a matrix. M is the co-occurrence matrix of the vocabulary V if $\forall w_i, w_j \in V$, $M_{i,j} = 1$ if $w_i \in C(w_j)$, otherwise $M_{i,j} = 0$.*

The easiest way to build such vectors is to use the rows of the co-occurrence matrix. However, the dimensionality increases linearly with the size of the vocabulary. Moreover this kind of matrix would be really sparse, because the majority of words appears rarely and therefore co-occurs with little part of the vocabulary. An intermediate step of dimensionality reduction can make interpretation easier and avoid the curse of dimensionality. However, calculate singular values of such a large matrix (more than one hundred thousands in the english vocabulary) is slow and any modification implies repeating the whole process. A better solution is to directly learn lower dimension vectors with a neural network, as it is proposed by Bengio *et al.* [32] (vectors were about 30 to 100 dimensions and the vocabulary was about 17000 words). The architecture of this model is shown in figure II.2. The vectors are by-products of a language modeling task: in this paper, the task is to predict a word in a corpus, given the n preceding words (its context). We search θ that maximizes the objective function O:

$$O = \frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid w_{t-1}, ..., w_{t-n}; \theta)$$

θ represents the parameters of the neural network (mainly weights and biases). P is an approximated probability calculated with the neural network. If we consider that $(w_t, w_{t_1}, ..., w_{t_n})$ is a frame of the corpus (a word and its context) then T is the number of frames. If we use a sliding window to get the frames, then T is also the number of words in the corpus. To ensure that all the probabilities sum to one, the output layer performs a softmax:

$$P(w_t \mid w_{t-1}, ..., w_{t-n-1}; \theta) = \frac{\exp(y_{w_t})}{\sum_{w \in V} \exp(y_w)}$$

where y_w is the unnormalized log-probability for each word of the vocabulary. y_w is a component of the vector y which expresses as follows:

$$y = b + W \cdot x + U \cdot h$$

W and U are the matrix of weights applied respectively to the output of projection layer x and hidden layer h. It appears that word vectors with characteristics discussed above ($\phi(w)$) can be extracted from the input layer.

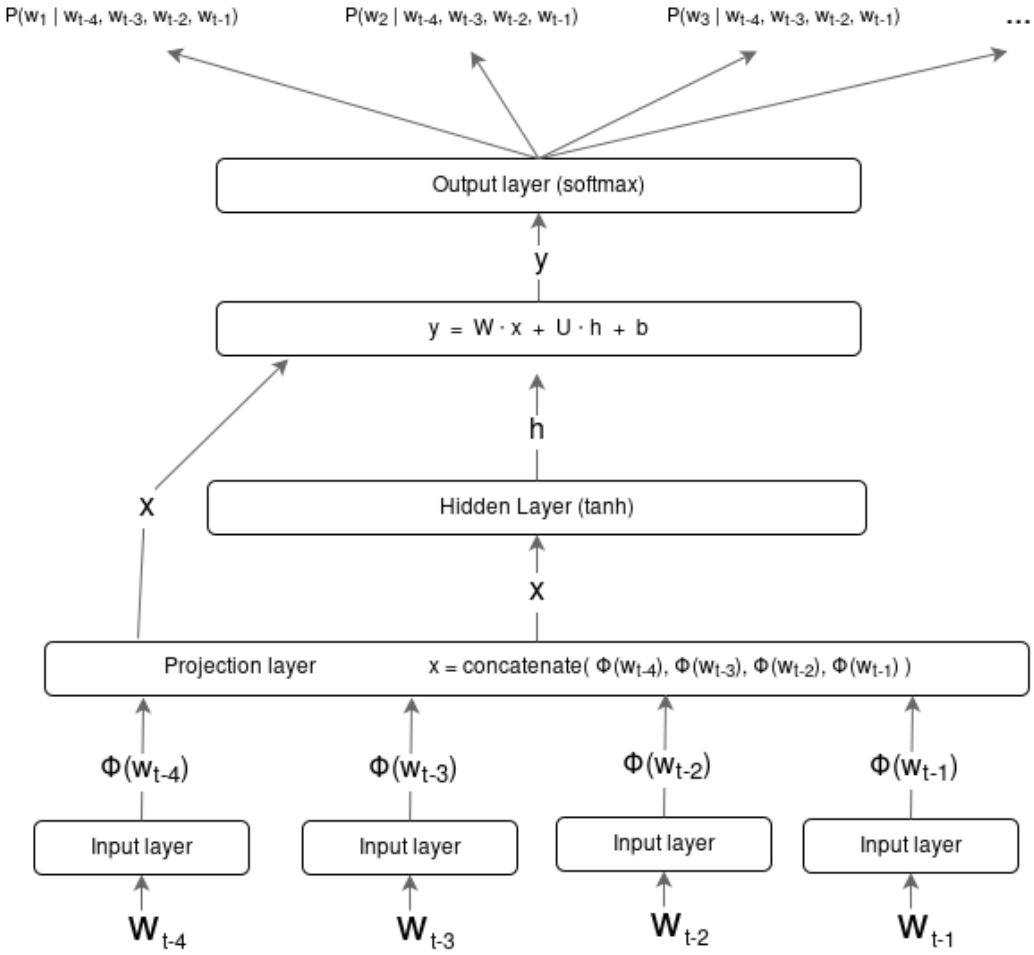


Figure II.2: The probabilistic feed forward neural network language model is composed of the input, projection, hidden, and output layers [32]. $\phi(w_t)$ is the feature vector of w_t.

According to Mikolov *et al.* it seems that the non linear hidden layer brings too much complexity [26]. In this paper, the author propose two shallow neural networks without hidden layer, Continuous Bag of Words (CBOW) and Skip-Gram model. This simplification allows the models to learn features from much bigger data sets.

II.2.2 Continuous Bag of Words

The Continuous Bag of Words (CBOW) model aims to predict a word in a corpus, given a window context (n words preceding and following this word). The architecture of this model is shown in figure II.3. O is the objective function to maximize :

$$O = \frac{1}{T}\sum_{t=1}^{T} \log P(w_t \mid w_{t+n}, \ldots, w_{t+1}, w_{t-1}, \ldots, w_{t-n}; \theta)$$

Compared to the previous model, the softmax layer is the same, except that the unnormalized log-probabilities y are now:

$$y = Wx$$
$$x = \frac{1}{|C(w_t)|} \sum_{w \in C(w_t)} \phi(w)$$

We can note there is no more hidden layer. The order of the words does not affect the result anymore because of the sum operation (instead of the concatenation in the previous model). That explains the name “Continuous Bag Of Words”.

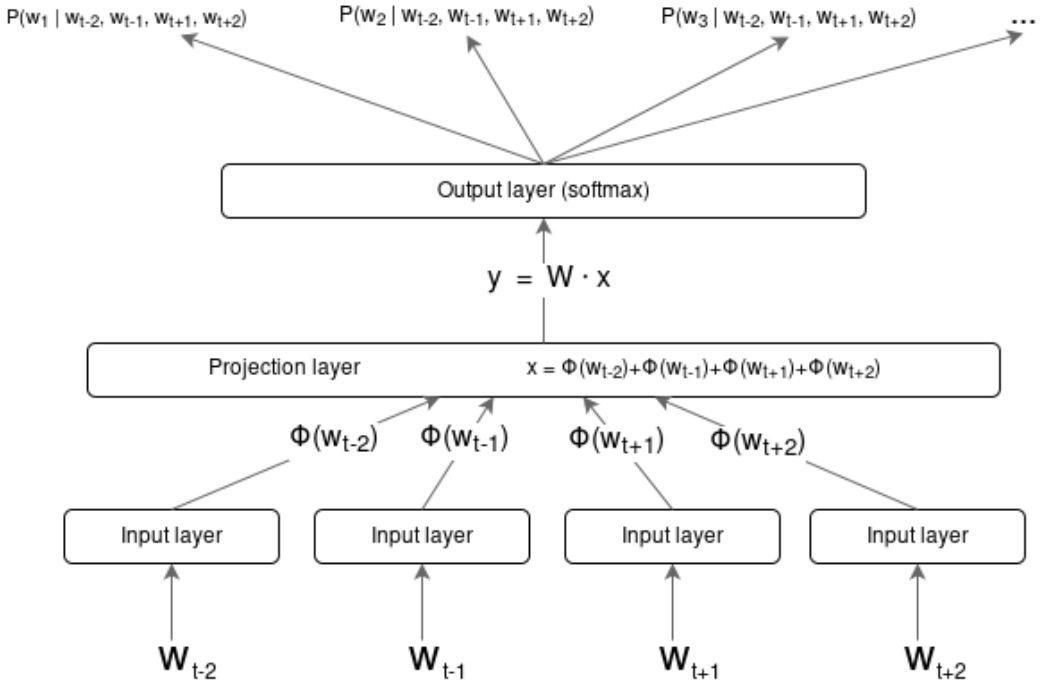


Figure II.3: Architecture of the CBOW model. CBOW learns to predict a word from its context. We can note the absence of any hidden layer.

II.2.3 Skip Gram

The Skip-gram model aims to learn the context of a word. The architecture of this model is shown in figure II.4. The objective is to maximize O:

$$O = \frac{1}{T}\sum_{t=1}^{T}\sum_{w_c \in C(w_t)} \log P(w_c \mid w_t; \theta)$$

Similarly to the CBOW, the softmax layer is still the same, but the unnormalized log-probabilities vector y is directly the product between W, the weights of the projection layer, and $\phi(w_t)$. Hence $P(w_c \mid w_t)$ can be expressed as follows [24]:

$$P(w_c \mid w_t) = \frac{\exp(W_{w_c}^T \cdot \phi(w_t))}{\sum_{w_i \in V} \exp(W_{w_i}^T \cdot \phi(w_t))}$$

We note W_{w_i} the row of W corresponding to the word w_i.

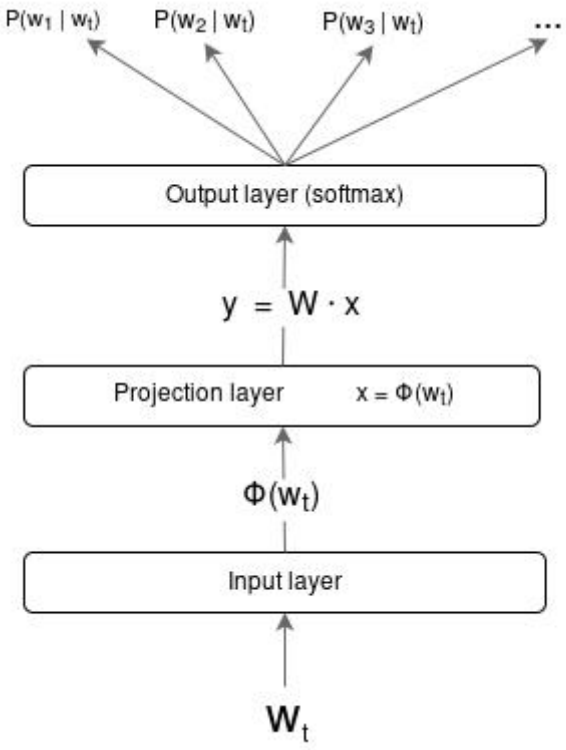


Figure II.4: Architecture of the Skip-Gram model. Skip-Gram learns to predict the words that belongs to the context of a target word. Similarly to the CBOW model, there is no hidden layer.

II.3 Efficient computation of a Softmax

The main issue with the Softmax layer is the normalization term:

$$\sum_{w_i \in V} \exp(W_{w_i}^T \cdot \phi(w_t))$$

It requires the calculation of a scalar product for each word in the vocabulary, even if the training model just needs $P(w_c \mid w_t)$ (Skip-gram model) to perform the backpropagation. Hierarchical Softmax and Negative sampling are two strategies that approximate efficiently a regular Softmax.

II.3.1 Hierarchical Softmax

As mentioned in [24], the Hierarchical Softmax has been first introduced by Morin and Bengio [33]. All the nodes of the neural network are not directly tied to the output layer anymore. They are now part of a binary tree, such that the number of nodes to evaluate in a training session can be proportional to $\log_2(|V|)$ as it can be seen on figure II.5 ($|V|$ is the size of the vocabulary). Given $\phi(w)$ each node produces two probabilities for going to each of the two children nodes (we call them *plain* and *dashed* referring to the lines in figure II.5). The probability $P(w \mid w_t)$ is the probability of going from the root node to the corresponding leaf node. The two probabilities given by a node sums to one, it guarantees that all the probabilities $P(w \mid w_t)$ also sums to one. If we follow the example in figure II.5, we define $P(w_2 \mid w_t)$ as follows :

$$\begin{aligned} P(w_2 \mid w_t) &= P(plain \mid N1, w_t) \times P(plain \mid N2, w_t) \times P(dashed \mid N4, w_t) \\ &= P(plain \mid N1, w_t) \times P(plain \mid N2, w_t) \times (1 - P(plain \mid N4, w_t) \\ &= \sigma(W_{N1}^T \cdot \phi(w_t) + b_1) \times \sigma(W_{N2}^T \cdot \phi(w_t) + b_2) \times (1 - \sigma(W_{N4}^T \cdot \phi(w_t) + b_4)) \end{aligned}$$

W_N is the weight vector for a node N. The nodes use an activation function σ whose result can be interpreted as a probability (such as sigmoid function). Mikolov *et al.* showed that using a

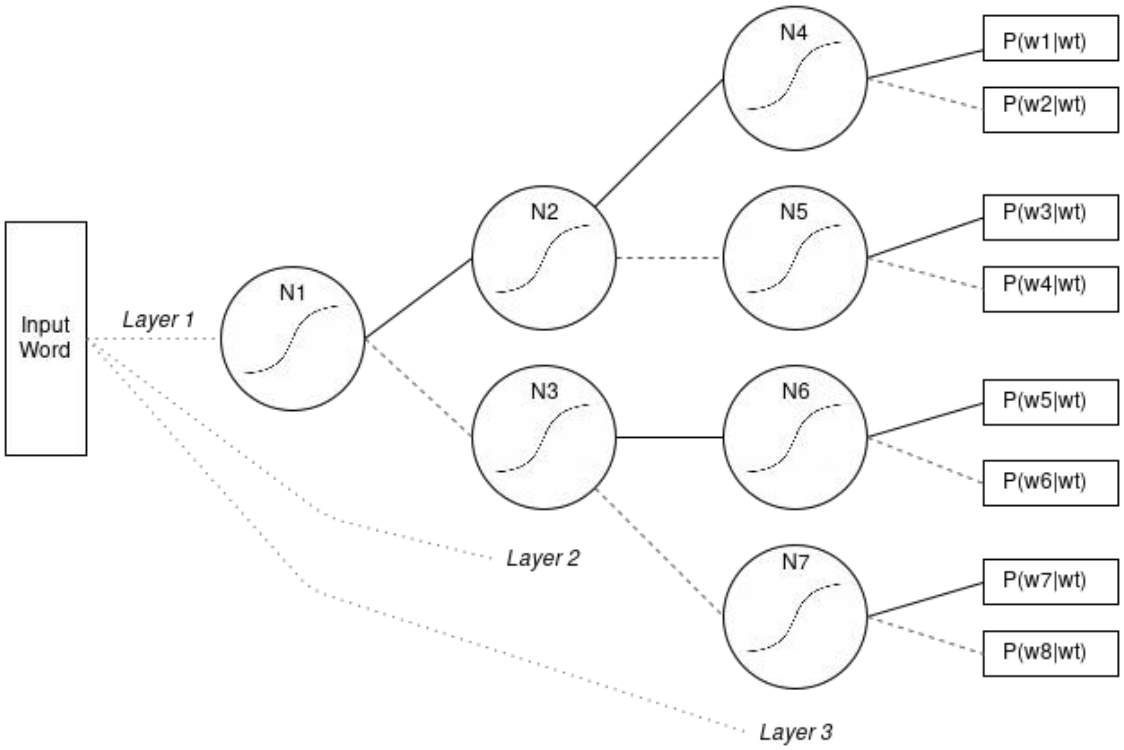


Figure II.5: The Hierarchical Softmax uses a binary tree structure to approximate the Softmax probabilities.

Huffman tree [34] (which places most frequent words closer from the root) can speed-up significantly the training session [24].

II.3.2 Negative Sampling

Negative sampling (NS) is a simplified version of Noise-contrastive estimation (NCE) [35]. They both are sampling approaches : instead of calculating a cross product for each word in the vocabulary, they just use samples of the vocabulary set. Unlike the hierarchical softmax, sampling approaches do not exhibit a softmax layer. Sampling approaches are only useful during the training time. A full softmax must be computed during the evaluation (that is not a problem since we do not use the model after the training, we just get the feature vectors). Instead of maximizing the log probability of the softmax, NS defines a new objective function [24, 36]:

$$\arg\max_{\theta} \sum_{(w_t,w_c)\in D} \log P(w_c \mid w_t;\theta) + \sum_{(w_1,w_2)\in D'} \log(1 - P(w_1 \mid w_2;\theta))$$

D is a set of words pairs (w_t, w_c) where w_c is drawn from the context of w_t in the corpus. D' is composed of random pairs words from the corpus. D' is called *the noise*. The goal is to jointly maximize $P(w_c|w_t)$ and minimize $P(w_1|w_2)$. By choosing the words w_1 and w_2 at random, we can expect that the most of the time w_1 is not in the context of w_2 (that explains the name *Negative sampling*). In a few words, given a couple of words (w_x, w_y) NS evaluates the probability that it comes from D. Negative sampling produces better word representations than hierarchical softmax (organized with a Huffman tree) in the same amount of time [24]. The reasons of this efficiency were not really understood but Goldberg and Levy showed later that SGNS (Skip gram with Negative sampling) is "implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information" [37]. Mutual information tells how likely two words are in each other context. That is why this sampling-based method is accurate, if we refer to the initial goal (directly learning a lower dimension matrix of words co-occurrence).

II.4 Why this approach is efficient for graph embedding

The size of social network and their evolution speed is the main hurdle for graph embedding algorithms. That is why parallelizability and distributability is also important. The algorithms and models introduced in this chapter exhibits such features and are this way, quite efficient for embedding big social graphs.

Scalability

The best asset for neural networks models such as CBOW and SG is that they do not need the whole training data since the beginning of the learning phase. They improve by reading the data step by step and correcting continuously the word features. That means whenever the training data grows, the model just have to compute the additional data to improve the features. That is really useful if one wants to keep his social graph embedding updated (social graph are known to grow continuously, at a very fast pace).

Parallelizability

These two models have a straightforward distributed implementation. Mikolov *et al.* used DistBelief (now part of TensorFlow[1]) [38] to run both of them in their experiments [26]. Distbelief is a framework that has been designed to run neural networks on thousands of machines. Perozzi *et al.* proved that using Asynchronous SGD [39] (Stochastic Gradient Descent [2], implemented in DistBelief) reduces significantly the running time without altering performances (in graph embedding context) [28]. Asynchronous SGD is used to perform a parallelized SGD. "Asynchronous" means there are risks of rewriting if two parallel implementations modify the same vectors at the same time (for example $\phi(w)$). As can be seen in figure II.1, many vertices have a narrow influence since they are not visited a lot. That minimizes the risk of rewriting while performing ASGD [39].

[1] https://www.tensorflow.org/about/ Tensorflow is an open-source framework for computation that in particular allows easy neural network implementations.

[2] Stochastic Gradient Descent (SGD) is a classical algorithm to search for optimum of an objective function.

Chapter III

Graph embedding algorithms

In the last chapter, we have introduced the mathematical models and tools on which the latest algorithms to learn social representation of nodes of a network are based. In this chapter, we'll consider main graph embedding algorithms, focusing on those based on machine learning tools. These algorithms produce vector representation that allow prediction, recommendation, *etc.*. We will first introduce interesting criterias for the evaluation of such algorithms. Then, we will talk about the three main approaches developed in the literature. Finally, we will tackle the issues of experimentation (data sets, methodology and good practices) and evaluation of these algorithms.

III.1 Evaluation criteria of graph embedding algorithms

Before getting down to the details of graph embedding algorithms, we need to set some criterias for their evaluation. The algorithm should obviously have a low complexity as we deal with large networks (with millions of nodes). It would be even better if it was parallelizable and easily distributable on a cluster of machines (vertically and horizontally scalable). The main question is how to evaluate graph embedding results. A good embedding should not suffer a loss of information compared to the original graph. Three kinds of tests are performed by the authors of embedding algorithms in III.2: classification, link prediction, and two dimensional projection.

Classification If we know the community decomposition of the network, we can try to obtain the same decomposition with classification or clustering algorithms. This could be applied with other properties we are able to label the nodes with. One-vs-rest logistic regression [13] was used to evaluate DeepWalk, node2vec and Line [28, 40, 41]

Link prediction We can remove a percentage of edges randomly of the network and try to predict the removed edges from the remaining ones.

Two dimensional projection We can compress a n-dimensions vector into a two dimensions one using any dimension reduction algorithm such as Principal Component Analysis and then look how the relative positions of nodes were preserved, if the nodes belonging to same communities are gathered or not. One advantage of this experiment is that it is quite easy to visualize and this way evaluate the quality of the result.

We can find other types of evaluation in the literature :

Estimations of shortest paths Berchenko *et al.* [42] decided to estimate the size of all the shortest paths (from one node chosen at random to the others) using the embedding by computing a score from the scalar product of two nodes embeddings. Then they compare the estimations with the real sizes. The error distribution is a measure for the quality of the algorithm.

Comparison with the Kleinberg model The Kleinberg model [43] produces jointly a social graph (with small-word properties) and its embedding in $\mathbb{Z}^2$. Kermarrec *et al.* [44] applied their own embedding algorithm on this social graph to compare it with the Kleinberg's embedding. They compare the two embeddings by computing the Pearson correlation between all node pairs' distances (the higher the correlation, the better the embedding is).

Apart from performance criteria, the vector representations should own some properties to facilitate their utilization. Perozzi *et al* present [28] (along with a graph embedding algorithm we'll introduce later) [28] four properties to preserve in the representation:

- Adaptability to the constant changes of the real social networks.
- The community structure of the network. The distance between to vectors should correspond to the similarity[1] between the two actors.
- Low dimensionality to avoid the curse of dimensionality [18]
- Vectors should be continuous to allow partial community membership and further classification studies.

III.2 Major approaches and algorithms to learn social representations

III.2.1 DeepWalk

DeepWalk was introduced by Perozzi *et al.* in 2014 [28]. The authors brought the idea of using recent work in word embedding. They emphasize interesting similarities between NLP and SNA (as seen in II.1). The main idea is to give *sentences* of nodes (instead of words) as input for word embedding algorithms. They used random walk to produce such *sentences* out of an unweighted graph (the overall architecture of Deepwalk can be seen in figure III.5). This way, it tries to capture the structure of the social representations with low dimensional vectors, as it was previously studied by Rosvall *et al.* [45]. DeepWalk is built on Skip-Gram [26] (see II.2.3) and Hierarchical Softmax [33] (see II.3.1). We will now focus on how DeepWalk creates sentences of nodes.

Definition 6 *Let be a graph* $G = (V, E)$ *and* $v_i \in V$. $W = (W^1_{v_i}, W^2_{v_i}, ..., W^t_{v_i})$ *is a random walk on graph* G *of* t *steps starting at node* $W^1_{v_i} = v_i$ *and* $\forall j \in [2, t]$, *step* $W^j_{v_i}$ *is chosen randomly among the adjacent nodes of* $W^{j-1}_{v_i}$.

The method used by DeepWalk is known as *truncated random walk* [28]. For each node (at step k) in the random walk, we produce a sentence of size $2w + 1$ where w is the window size, that is to say the number of nodes before and after the root of the walk in the sentence ($W^{k-w}_{v_i}$, ..., $W^k_{v_i}$, ..., $W^{k+w}_{v_i}$). Skip-Gram uses directly these sequences to learn the node features. DeepWalk performs $\gamma \times |V|$ random walks such that each node is the root of γ random walks. Each random walk is independent of each other, hence this task can be easily parallelized. According to the authors, "most of the benefit is accomplished by starting $\gamma = 30$ walks". A summary of the complete algorithm can be consulted below (figure III.1).

[1] The notion of similarity can have different interpretations in a social network. Hence there are many ways to calculate it. For example, it can be measured with the weight on the edges or with the number of common adjacent nodes.

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

```
Initialization: Sample Φ from U^{|V|×d}
Build a binary Tree T from V
for i = 0 to γ do
    O = Shuffle(V)
    for each v_i ∈ O do
        W_{v_i} = RandomWalk(G, v_i, t)
        SkipGram(Φ, W_{v_i}, w)
    end for
end for
```

Figure III.1: The DeepWalk algorithm as presented by its authors Perozzi *et al.* [28]

Cao *et al.* have recently proposed *random surfing* [46], an amelioration of the *truncated random walk*. According to the authors, it aims to correct two issues. First, the finite length of the sentences generated by DeepWalk creates bias at the boundaries, where nodes context is not well captured. Secondly, the parameters (number and length of the walks, t and γ) are not easy to determine. The authors argue that *random surfing* produces node embedding with properties much closer to Word2vec [24] embedding than DeepWalk [28].

III.2.2 LINE

LINE was introduced by Tang *et al.* [41] and was partly inspired by the concept of graph factorization from Ahmed *et al.* [47]. The algorithm is designed for directed graphs and works with positive weights on the edges. The objective of LINE is to compute an embedding which best preserves the structure of the graph. To reach this goal, the authors defined the notions of first and second order proximity. These are local properties expressing how similar are two nodes in the network.

The first order proximity of two nodes is materialized by the edge between them. If there is no edge, the two nodes are not similar with regard to the first order proximity. If the edge is weighted, we can quantify the first order proximity. According to Jian Tang *et al.*, the Laplacian Eigenmaps algorithm [21] could already preserves some kind of first order of proximity [41]. The authors argue that considering only the first order proximity is not sufficient to capture a network structure. If a node shares many neighbors with another, they are quite similar, even if they are not connected, this is the second order proximity. That is the idea behind Word2Vec: similar words tends to appear in similar context. Since it is built on top of a Word2Vec model, DeepWalk preserves the second order of proximity. However, Tang *et al.* reproached DeepWalk for not having an objective clear enough. By contrast with it, they introduced two objective functions for LINE, one for each order of proximity.

In order to model the first order priority objective, the authors defined the joint probability for an undirected edge (i, j)[2] between the vertices v_i and v_j as follows:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-u_i^T \cdot u_j)}$$

where u_i and u_j are the embeddings of nodes v_i and v_j. This equation defined a target probability distribution $p_1(\cdot, \cdot)$ on the space $V \times V$. Empirical probabilities toward which this target distribution

[2]The first order proximity as defined by the authors only applies on undirected edges, that is not the case for second order proximity as we will see later. That is why we told earlier that the algorithm was only relevant for undirected networks.

must tends can be found by dividing empirical weight $w_{i,j}$ between nodes v_i and v_j by the sum of all weights in the network. The objective function is then to minimize the distance between the two distributions:

$$O_1 = \text{dist}(\hat{p}_1(\cdot,\cdot), p_1(\cdot,\cdot))$$

which the authors simplify using the Kullback-Leibler divergence formula [48] in:

$$O_1 = -\sum_{(i,j)\in E} w_{i,j} \log p_1(v_i, v_j)$$

The second order proximity defines the similarity between two nodes following the number of connections they share. Therefore a notion of context comes into play. Hence the authors defined a context vector representations for each node. It is noted u'_i for the vertex v_i. The standard vector representation is still u_i for vertex v_i. For each directed edge (i, j) of the network, they define the probability that v_i belongs to the context of v_j:

$$p_2(v_j|v_i) = \frac{\exp(u_j'^T \cdot u_i)}{\sum_{k=1}^{|V|} \exp(u_k'^T \cdot u_i)}$$

Using a similar technique as for first order proximity they define:

$$O_2 = -\sum_{(i,j)\in E} w_{i,j} \log p_2(v_i|v_j)$$

where $w_{i,j}$ is the second order proximity. $w_{i,j}$ is obtained by calculating the similarity between the two vectors of first order proximity values with all other nodes for each two nodes.

To combine first and second order proximities, the two vectors obtained with the two objective functions are concatenated. Thus, LINE tries not only to preserve the local structure of the network but also the global structure. The computation being quite expensive (especially for the second objective function), the authors propose to use approaches evoked earlier in this document such as Negative Sampling. According to the authors, LINE possesses some advantages over DeepWalk. First, it provides a clear objective function that ensures properties (especially first and second order proximity) of the network are preserved. Second, the algorithm is applicable to both weighted and unweighted network[3].

III.2.3 node2vec

node2vec is an algorithm introduced by Grover *et al.* in 2016 [40]. node2vec tries to push further the concept of random walk introduced with DeepWalk [28] by introducing *biased random walk* (the overall architecture of node2vec can be seen in figure III.5). It works for both directed/undirected and weighted/unweighted networks.

According to the authors, there are two main strategies to browse a graph from a node: Breadth-First Search (BFS)[4] and Depth-First Search (DFS)[5]. We can apply the same strategies to explore and sample the neighborhood of a node (similarly to a random walk) and observations made by the authors show that they do not produce embedding with the same properties. They distinguish to kind of similarities between the nodes:

Homophily The nodes that are highly connected and in the same communities.

Structural equivalence The nodes that play a similar role (hubs for example).

BF sampling seems to produce an embedding that preserves structural equivalence, whereas DF sampling seems to preserve homophily. The goal of the authors is then to build a random walk synthesizing both sampling strategies.

They therefore designed a random walk where the transitions probabilities from step k to step $k + 1$ depends also on the node visited in step $k - 1$. The authors introduced two parameters to control the sampling strategy. If the walk just went from node t to node v, it must now select an

[3] Indeed DeepWalk does not use the weights on the edges.
[4] one explores first the closer neighborhood of the node.
[5] one explores first as far as possible starting from the node before coming back.

adjacent node x of v to visit (it can be t again). The probability of transition from node v to node x relied by an edge weighted by w_{vx} is defined by:

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$$

where

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} \text{ if } d_{tx}=0 \\ 1 \text{ if } d_{tx}=1 \\ \frac{1}{q} \text{ if } d_{tx}=2 \end{cases}$$

d_{tx} is the shortest path between nodes t and x. p and q are the two parameters introduced by the authors, as can be seen in figure III.2. They both control "how fast the walk explores and leaves the neighborhood" of the root node. p is called *return parameter*, it controls the probability to go back to node t. q is the *in-out parameter*, it controls the probability to visit a node that is not connected to t and therefore to explore potential new parts of the network. In a toy example, Grover *et al.* used the values ($p = 1$, $q = 0.5$) to preserve homophily (DFS walk) and the values ($p = 1$, $q = 2$) to preserve structural equivalences (BFS walk).

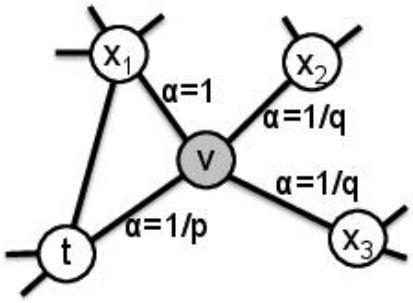


Figure III.2: The biased random walk introduced by node2vec [40]. Each target node has a different bias α according to its distance to t, the node visited at the previous step.

After generating these walks, node2vec uses Skip-gram with Negative Sampling (respectively introduced in II.2.3 and II.3.2) to obtain vector representations of the nodes. A summary of the complete algorithm can be consulted below (figure III.3).

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  $G' = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter = 1$ **to** r **do**
    for all nodes $u \in V$ **do**
      $walk$ = node2vecWalk(G', u, l)
      Append $walk$ to $walks$
  f = StochasticGradientDescent(k, d, $walks$)
  return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
  Inititalize $walk$ to $[u]$
  for $walk\_iter = 1$ **to** l **do**
    $curr = walk[-1]$
    V_{curr} = GetNeighbors($curr, G'$)
    s = AliasSample(V_{curr}, π)
    Append s to $walk$
  return $walk$

Figure III.3: The node2vec algorithm as presented by its authors Grover *et al.* [40]

Furthermore, node2vec seems to scale linearly with the number of nodes (figure III.4). In this experiment, node2vec has been run in parallel, using asynchronous stochastic gradient descent algorithm to optimize the objective function [39].

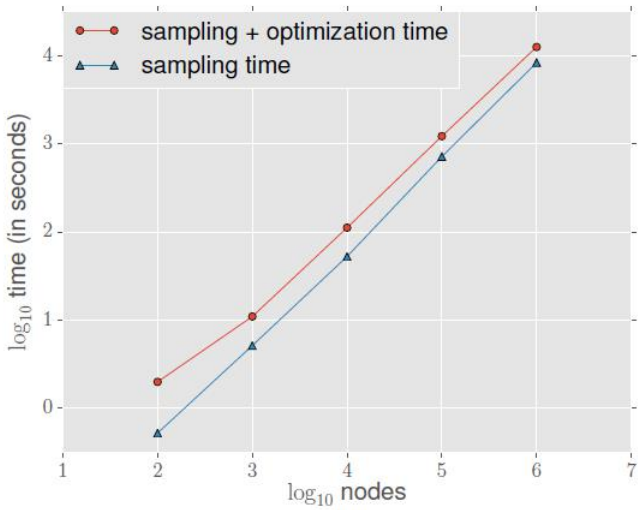


Figure III.4: Scalability results of the node2vec algorithm

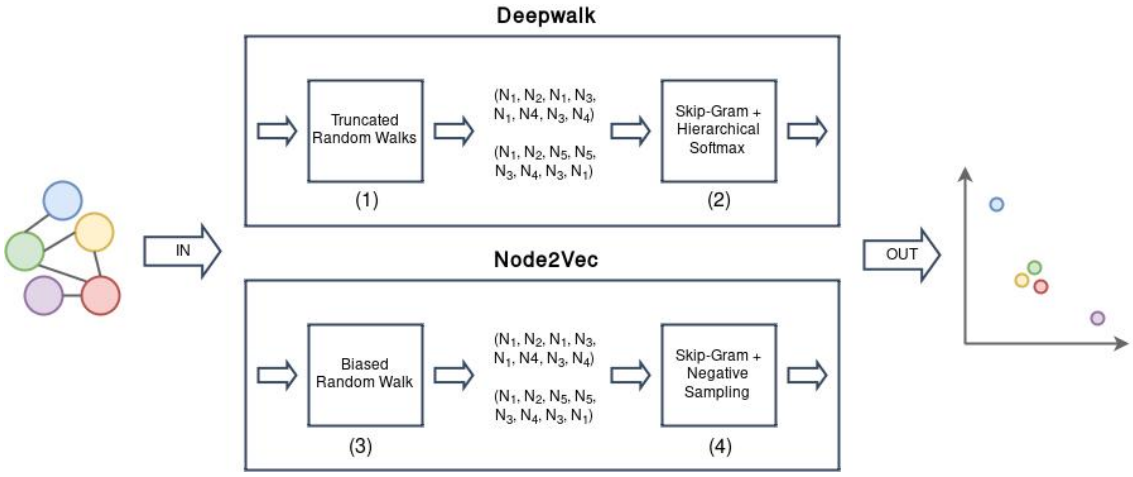


Figure III.5: Deewpalk and node2vec exhibit the same overall architecture. They both use a random walk algorithm to produce "sentences of nodes" (1, 3 in the figure). The "sentences" are then used as input data for a word embedding algorithm (2,4 in the figure). The output are the graph embedding vectors.

III.3 Experimentation

In this section, we tackle some experimentation observations we can make from the articles. First, the choice of data sets is primordial. In the paper introducing node2vec [40], the authors used three differents data sets for each experiment. Even if they are quite big, we could argue that the size of the data sets are not spaced enough (they range from 10^3 to 10^4 nodes) and more variety could lead to different results. The authors of LINE use much larger data sets for experimentation, around 10^6 nodes.
One of the main problems addressed by graph embedding algorithms is nodes classification. When assessing the results of a classification (binary or multilabel), all articles present F_1-score (or F measure). Classic formulation of F_1 score for binary classification is:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

When working with several experiments, one must calculate the average precision and recall to get the F_1 score. There is two ways to do so: micro average and macro average. They lead to different results even if they represent the same thing. Micro average is calculated using individual true positives, false negatives *etc*, summing them and then make the results. Macro average directly make the average of the precision and recall found on different experiments. For example, we calculate the average precision of two experiments using first experiment precision P_1, true positives TP_1, false positives FP_1 and symmetrically for second experiment:

$$\text{micro precision} = \frac{\text{TP}_1 + \text{TP}_2}{\text{TP}_1 + \text{TP}_2 + \text{FP}_1 + \text{FP}_2}$$
$$\text{macro precision} = \frac{P_1 + P_2}{2}$$

When using more than two classes for classification, one can calculate the precision and recall for one class against the others and then make an average for all classes. To go further, an article written by Sokolova *et al.* [49] present a review of classic performance measures for classification tasks.

III.4 Comparison and results

In this section, we present results from different experiments for the three algorithms we introduced above: DeepWalk, LINE and node2vec. As the most recent, the article presenting node2vec [40]

provides comparison of the three. The authors performs two tests: multilabel nodes classification and link prediction.
The multilabel classification is performed on three different data sets presented below.

data set	number of nodes	number of edges	number of labels
BlogCatalog	10312	333983	39
PPI	3890	76584	50
Wikipedia	4777	184812	40

The results are presented below (figures from the article [40]):

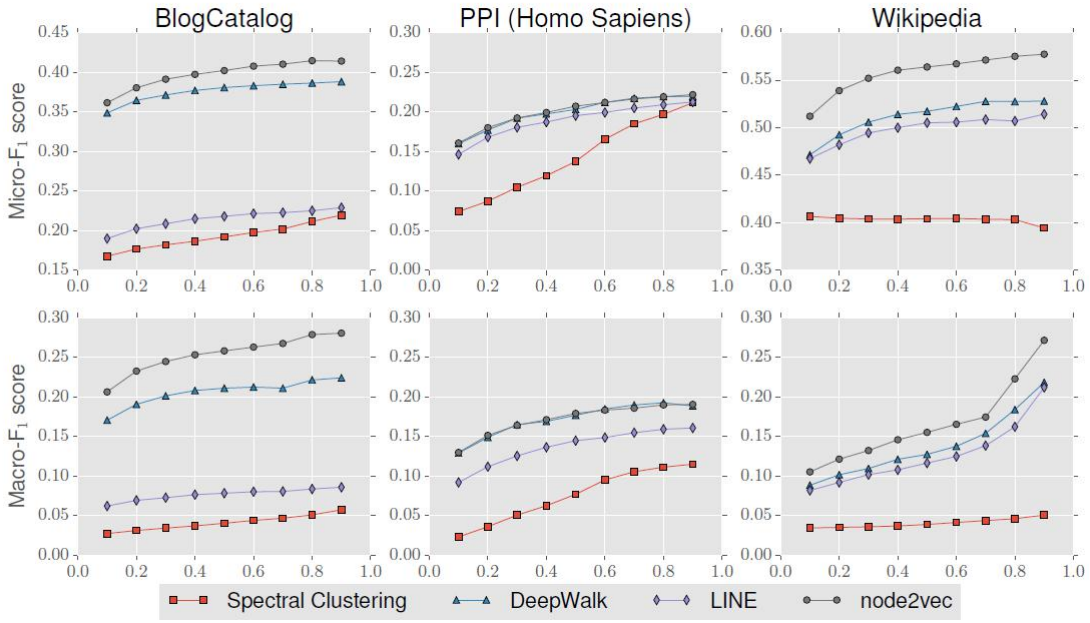


Figure III.6: Results for the multilabel classification experiment

On these figures, we can see clearly that node2vec seems above other approaches, DeepWalk got close results on the PPI data set, LINE is really behind on the BlogCatalog data set. However, without the score obtained by a random classifier on the same problem, this is impossible for the reader to have an idea of the absolute performances of the algorithms. Calculations we made on a random classifier choosing equiprobably between forty labels[6]. We got a F_1-score of about 0.025, in other words, ten times lower than the results obtain by the authors of node2vec in their experiment.
The second test is about link prediction, this is, according to the authors, the first time feature learning algorithms are tested on node prediction and they choose to confront machine learning algorithms with other classical tools of social network analysis such as spectral clustering and Jaccard's coefficient. data sets used for the experiment are presented below.

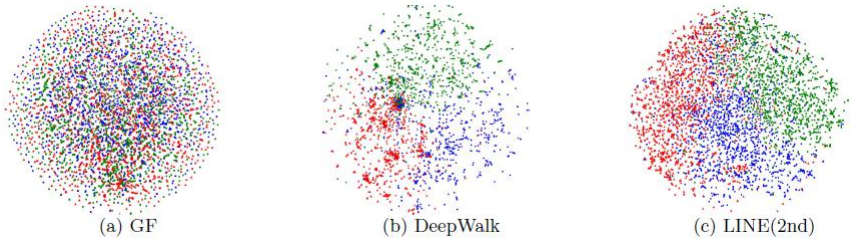
data set	number of nodes	number of edges
Facebook	4039	88384
PPI	19706	390633
ArXiv	18722	198110

This time, the results are more interesting since all the algorithms got honorable results: the authors give the area under curve. Here again, node2vec seems to perform best, quickly followed by

[6]We highly simplified the test. In the article, the classifier can assign many label to one node. However, even with simpler hypothesis we should obtain a good estimation for the F_1-score, at least an upper born.

DeepWalk. We remark a huge drop of performance of each algorithm on the PPI data set that the authors do not give explanations to.

However, the authors of LINE also present results [41], comparing with deepwalk. They tested their algorithm for two tasks on social network analysis: nodes classification and two-dimensional visualization. They used five different networks of three types: two social networks (Flickr and YouTube, the same used for experimentations in DeepWalk article), two citation networks and a language network created from Wikipedia. The authors insist on the size (around a million of nodes) and on the variety of the networks they used, different origins but also different features (directed/undirected, weighted/unweighted).
Node classification was operated on Wikipedia pages and LINE [41] outperformed DeepWalk for a few percents. The second experiment consists in comparing the two-dimensional visualization created from the embedding. The authors compared three algorithms: DeepWalk [28], Graph factorization [47] and LINE using the second-order proximity. They work on a citation network: DBLP authors citation which is directed and weighted and have about 500000 nodes and 20 millions edges. The visualization is presented below, the colors indicate the research field of the authors:



(a) GF (b) DeepWalk (c) LINE(2nd)

Figure III.7: Two-dimensional projection of a citation network using embedding from three algorithms

Graph factorization produces a representation that mix up the communities. DeepWalk's representation shows more clearly the community structure but we observe gathering points which the authors interpret as agglomeration of points with multiple labels. LINE (with the second order proximity), according to the authors, produces the most satisfying figure.

From these results, we can conclude that the approach of LINE [41], quite different from DeepWalk [28] and node2vec [40], owns some assets on the others for few tasks such as two-dimensional projection of the network.

Chapter IV

Conclusion and perspectives

IV.1 Conclusion

In this state of the art, we introduced main approaches to produce social vector representations from vertices of a social network. These approaches, based on artificial neural networks, can be divided into two categories. The first leans on natural language processing tools and skip-gram model (DeepWalk [28], node2vec [40]) and try to sample the neighborhood of a node using random walks. The other approach is proposed by the authors of LINE [41] who defined the objective of their algorithm in order to preserve two orders of proximity.
These algorithms have been tested on classical problems of social network analysis : node classification and link prediction. They have produced (especially node2vec) good results on traditional directed and weighted networks. In the last sections of this report, we would like to tackle some new perspectives and further issues for graph embedding research embedding of more complex networks, community embedding and the issue of producing lower dimension embedding.

IV.2 Bipartite graphs and social network modeling

Definition 7 *A bipartite graph, or two-mode graph, is a graph $G = (U, V, E)$ where U and V are two disjoint sets of vertices (sometimes called top and bottom) and E is the set of edges.*

Similarly to a classic graph, it can be weighted or not, directed or not. For example, we can associate people and films to make an weighted bipartite graph where the weight on an edge between a film and a person represent how much the person likes the film. It is possible to define k-partite graphs, with k the number of disjoint sets of vertices of the graph. As for classical networks, the questions of modeling and random generation arise that Latapy and Guillaume address in two articles [50, 51]. To them, bipartite graphs can be used to model complex networks such as author-articles networks and they list three principal features that must be preserved from real world bipartite graphs: high clustering, power law degree distribution and logarithmic average distance.

The question of embedding of a bipartite graph has not yet been tackled in the literature. However, we remark some idea and close approaches. Kunegis *et al.* succeeded in adapting their methodology to learn weights on edges and perform link prediction on bipartite graphs [52] but it does not use embedding approach. It is based on adjacency matrix transformation and eigenvalue decomposition which we told earlier is not quite adapted to study large networks. Egyed-Zsigmond *et al.* quoting Latapy *et al.* [50] precise in an article about tools of SNA [53] that it is possible to transform a bipartite graph into a classic graph by using projection on one set of vertices and aggregation. After such transformation, it could be possible to apply embedding algorithms we presented earlier, or at least, adaptations of them.

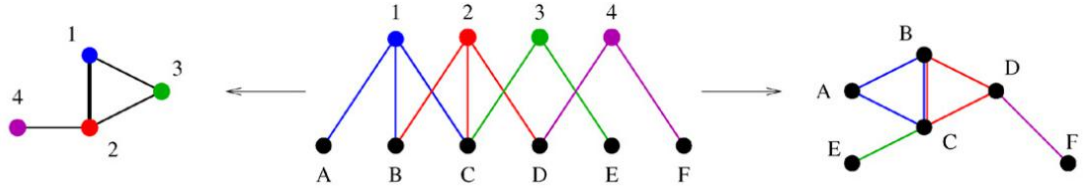


Figure IV.1: A figure taken from an article of Latapy *et al.* [54] showing how to project a two-mode graph on a one-mode graph.

IV.3 Community structure preservation

As said in I.1, communities preservation and detection is an interesting problem of SNA. However, the algorithms in III.2 only preserves the neighborhood of the node: the community structure is not considered while building the node representation. Zheng *et al.* introduced ComEmbed, an algorithm that learns jointly node embedding and community embedding [55] (community embedding assigns a vector for each community in the social graph). They argue that community embedding helps producing a better node embedding (that preserves communities) and that the reverse is also true: "community embedding and node embedding reinforce with each other" [55]. ComEmbed relies on the same two objective functions LINE uses for preserving the first and the second order of proximity (see III.2.2). In order to maintain community structures, the authors introduced a third order of proximity: two nodes are similar if they belong to the same community. Zheng *et al.* used a new evaluation test in order to measure the community preservation (along with node classification, the standart test). Since ComEmbed promising results on the two tests, community embedding is a really interesting approach for graph embedding.

IV.4 Producing low dimension embedding

As said previously in this report, there are two major issues with high dimension representations. They are affected by the curse of dimensionality and they slow down computation. It is therefore interesting to reduce the dimension of the representation produced by graph embedding algorithm while keeping good performances on link prediction or node classification.
As also mentioned earlier, two-dimension vectors are practical for visualization. On this aspect, we show in the last chapter that LINE approach produces better results than DeepWalk, that is also true with ComEmbed as it can be noticed from the figure below [1]. Therefore, we can suppose that LINE and ComEmbed better preserve the structure of the network in low dimensions.

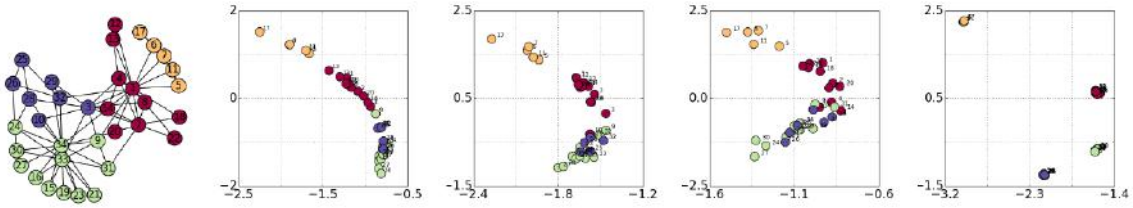


Figure IV.2: Result of experiment of two dimension projection on Karate Club data set for the four major algorithms we featured in this report. Figure from Zheng *et al.* [55], exact parameters used can be found in the article.

IV.5 Dimensionality reduction

However, another approach can be considered: reducing the dimension of the vector after creating them. Traditional dimensionality reduction algorithms could therefore be considered but they

[1] This experiment is similar to the one we presented in the section III.4 of the last chapter. It can also be compared to a figure found in DeepWalk article [28] consisting in a two dimension embedding of the karate club graph.

still bring the problems evoked in the introduction of this report. The authors of the previously presented *random surfing*, Cao *et al.* [46] used a denoising algorithm to compress the vectors. This kind of algorithms can be used to reduce the noise of data such as images. Cao *et al.* used an algorithm called stacked denoising autoencoder [56] to produce vector representations with the lowest number of dimensions possible.

Bibliography

[1] Alain Degenne and Michel Forsé. *Introducing social networks*. Sage, 1999.

[2] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[3] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, pages 425–443, 1969.

[4] Lei Tang and Huan Liu. Graph mining applications to social network analysis. In *Managing and Mining Graph Data*, pages 487–513. Springer, 2010.

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

[6] Adrien Guille, Hakim Hacid, Cecile Favre, and Djamel A Zighed. Information diffusion in online social networks: A survey. *ACM SIGMOD Record*, 42(2):17–28, 2013.

[7] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. Node classification in social networks. In *Social network data analytics*, pages 115–148. Springer, 2011.

[8] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

[9] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156. IEEE, 2013.

[10] Cody Buntain and Jennifer Golbeck. Identifying social roles in reddit using network structure. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 615–620, New York, NY, USA, 2014. ACM.

[11] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005.

[12] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proc. of ACM SIGKDD*, pages 1105–1114, 2016.

[13] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] Kelly H Zou, A James O'Malley, and Laura Mauri. Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models. *Circulation*, 115(5):654–657, 2007.

[16] Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Embedding graphs in books: A layout problem with applications to vlsi design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987.

[17] R. Bellman and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[18] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*, pages 257–258. Springer, 2011.

[19] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

[20] Bin Luo, Richard C Wilson, and Edwin R Hancock. Spectral embedding of graphs. *Pattern recognition*, 36(10):2213–2230, 2003.

[21] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[22] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[23] Wikipedia. Eigenvalue algorithm — wikipedia, the free encyclopedia, 2016. [Online; accessed 14-November-2016].

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[25] 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.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[27] Christopher D Manning, Hinrich Schütze, et al. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

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

[29] Paul Erdős and A Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[30] Steve Lawrence and C Lee Giles. Accessibility of information on the web. *Nature*, 400(6740):107–107, 1999.

[31] Peter J Kwantes. Using context to build semantics. *Psychonomic Bulletin & Review*, 12(4):703–710, 2005.

[32] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

[33] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[34] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[35] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012.

[36] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[37] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[38] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[39] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[40] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[41] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. ACM, 2015.

[42] Yakir Berchenko and Mina Teicher. Graph embedding through random walk for shortest paths problems. In *International Symposium on Stochastic Algorithms*, pages 127–140. Springer, 2009.

[43] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170. ACM, 2000.

[44] Anne-Marie Kermarrec, Vincent Leroy, and Gilles Trédan. Distributed social graph embedding. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1209–1214. ACM, 2011.

[45] Martin Rosvall and Carl T Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[46] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. 2016.

[47] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[48] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[49] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[50] Jean-Loup Guillaume and Matthieu Latapy. Bipartite structure of all complex networks. *Information processing letters*, 90(5):215–221, 2004.

[51] Jean-Loup Guillaume and Matthieu Latapy. Bipartite graphs as models of complex networks. *Physica A: Statistical Mechanics and its Applications*, 371(2):795–813, 2006.

[52] Jérôme Kunegis and Andreas Lommatzsch. Learning spectral graph transformations for link prediction. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 561–568. ACM, 2009.

[53] David Combe, Christine Largeron, Elod Egyed-Zsigmond, and Mathias Géry. A comparative study of social network analysis tools. In *Web Intelligence & Virtual Enterprises*, 2010.

[54] Matthieu Latapy, Clémence Magnien, and Nathalie Del Vecchio. Basic notions for the analysis of large two-mode networks. *Social networks*, 30(1):31–48, 2008.

[55] Vincent W Zheng, Sandro Cavallari, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. From node embedding to community embedding. *arXiv preprint arXiv:1610.09950*, 2016.

[56] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.