LEARNING NETWORK NODE REPRESENTATIONS FROM STRUCTURAL IDENTITY

Leonardo Filipe Rodrigues Ribeiro

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Junho de 2017

LEARNING NETWORK NODE REPRESENTATIONS FROM STRUCTURAL IDENTITY

Leonardo Filipe Rodrigues Ribeiro

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2017

Ribeiro, Leonardo Filipe Rodrigues

Learning Network Node Representations from Structural Identity/Leonardo Filipe Rodrigues Ribeiro. – Rio de Janeiro: UFRJ/COPPE, 2017.

XII, 67 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2017.

Referências Bibliográficas: p. 63 – 67.

1. Feature Learning. 2. Node Embeddings. 3. Structural Identity. I. Ratton Figueiredo, Daniel. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

“Não haverá borboletas se a vida
não passar por longas e
silenciosas metamorfoses.”
Rubem Alves

Agradecimentos

Agradeço aos meus pais, Andreisa e José Ribeiro, por sempre acreditarem no meu potencial e por, mesmo com todas as dificuldades, sempre terem apoiado meus estudos. Às minhas irmãs, Fernanda e Flávia, pelo apoio mesmo distante e por todo carinho dedicado a mim durante a vida. Agradeço também à Nathália Araujo, por sempre me incentivar a perseguir meus sonhos, e com paciência suportar toda tensão e ausência adivindas deste trabalho. Pelo consolo e afeto nos momentos de angústia, não esquecerei. Amo muito todos vocês.

Agradeço ao meu orientador acadêmico, Professor Daniel, por todo o apoio e conselhos durante a elaboração desse trabalho. Seu encantamento pela ciência e ensino me motivaram na jornada do mestrado. Deixo claro que seus ensinamentos ficarão para sempre em minha memória. Sem você, não teria conseguido.

Agradeço também ao Professor Felipe França, por me acolher no início do mestrado e me mostrar vários caminhos possíveis para minha trajetória acadêmica. Ao Pedro Savarese, pela grande atenção despendida no início dessa pesquisa e pelas ajudas constantes durante todo o trabalho. Ao amigo Jhonatan Oliveira, pelas produtivas conversas, desde os tempos de residência no Vale do Aço, bem como pela ajuda na revisão do presente texto.

Agradeço aos amigos que fiz durante esses dois anos na UFRJ e no LAND, por toda ajuda a mim concedida: David Coelho, Evandro Macedo, Giulio Iacobelli, Jefferson Simões, João Vitor, José de Paula, Leandro Santiago, Luiz Fernando, Paulo Nascimento, Raphael Erthal, Raul Barbosa, Renato Souza e Wladimir Cabral.

À Universidade Federal do Rio de Janeiro, e em especial ao Programa de Engenharia de Sistemas e Computação da COPPE, por ter contribuído para minha formação e ter recebido tão bem esse mineiro.

Ao professor Rodrigo Gaiba, que, desde os tempos da graduação, sempre me apoiou, acreditou e apostou no meu potencial. Compartilho essa conquista com você, amigo.

Aos colegas de turma e de trabalho, pelas discussões que ajudaram a evoluir e amadurecer meus conhecimentos.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APRENDIZADO DE REPRESENTAÇÕES LATENTES DE VÉRTICES EM REDES BASEADO EM IDENTIDADE ESTRUTURAL

Leonardo Filipe Rodrigues Ribeiro

Junho/2017

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Identidade estrutural é um conceito de simetria, no qual vértices em uma rede são identificados de acordo com a estrutura da rede e com seus relacionamentos com outros vértices. A identidade estrutural tem sido estudada na teoria e na prática durante as últimas décadas, mas, somente recentemente, técnicas para aprendizado de representações latentes vêm sendo utilizadas neste contexto. Este trabalho apresenta o *struc2vec*, um framework inovador e flexível, utilizado para o aprendizado de representações latentes da identidade estrutural de vértices. *struc2vec* usa uma hierarquia para medir a similaridade de vértices em diferentes escalas, e constrói um grafo multi-camadas para codificar similaridades estruturais e gerar contexto estrutural para vértices. Experimentos numéricos indicam que recentes técnicas para aprendizado de representações de vértices falham em capturar uma forte noção de identidade estrutural, enquanto *struc2vec* exibe um desempenho muito superior nestas tarefas, uma vez que supera as limitações das técnicas anteriores. Como consequência, experimentos númericos indicam ainda que *struc2vec* melhora o desempenho em tarefas de classificação que dependem mais da identidade estrutural.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

LEARNING NETWORK NODE REPRESENTATIONS FROM STRUCTURAL IDENTITY

Leonardo Filipe Rodrigues Ribeiro

June/2017

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

Structural identity is a concept of symmetry in which network nodes are identified according to the network structure and their relationship to other nodes. Structural identity has been studied in theory and practice over the past decades, but only recently has it been addressed with representational learning techniques. This work presents *struc2vec*, a novel and flexible framework for learning latent representations for the structural identity of nodes. *struc2vec* uses a hierarchy to measure node similarity at different scales, and constructs a multilayer graph to encode structural similarities and generate structural context for nodes. Numerical experiments indicate that state-of-the-art techniques for learning node representations fail in capturing stronger notions of structural identity, while *struc2vec* exhibits much superior performance in this task, as it overcomes limitations of prior approaches. As a consequence, numerical experiments indicate that *struc2vec* improves performance on classification tasks that depend more on structural identity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In almost all networks, nodes tend to have one or more functions that greatly determines their role in the system. For example, individuals in a social network have a social role or social position [1, 2], while proteins in a protein-protein interaction (PPI) network exert specific functions [3, 4]. Intuitively, different nodes in such networks may perform similar functions, such as interns in the social network of a corporation or catalysts in the PPI network of a cell. Thus, nodes can often be partitioned into equivalent classes with respect to their function in the network.

Although identification of such functions often leverage node and edge attributes, a more challenging and interesting scenario emerges when node function is defined solely by the network structure. In this context, not even the labels of the nodes matter but just their relationship to other nodes, represented by the edges. Indeed, mathematical sociologists have worked on this problem since the 1970s, defining and computing *structural identity* of individuals in social networks [1, 2, 5]. Beyond sociology, the role of webpages in the webgraph is another example of identity (in this case, hubs and authorities) emerging from the network structure, as defined by the celebrated work of Kleinberg [6].

The most common practical approaches to determine the structural identity of nodes are based on distances or recursions. In the former, a distance function that leverages the neighborhood of each node is used to measure the distance between all node pairs. Then clustering or matching is used to place nodes into equivalent classes [7, 8]. In the later, a recursion with respect to neighboring nodes is constructed and then iteratively unfolded until convergence, with final values used to determine the equivalent classes [6, 9–11]. In this work, we provide an alternative methodology, one based on unsupervised learning of representations that capture the structural identity of nodes.

Recent efforts in learning latent representations for nodes in networks have been quite successful in performing classification and prediction tasks [12–15]. In particular, these efforts encode nodes using as context a generalized notion of their

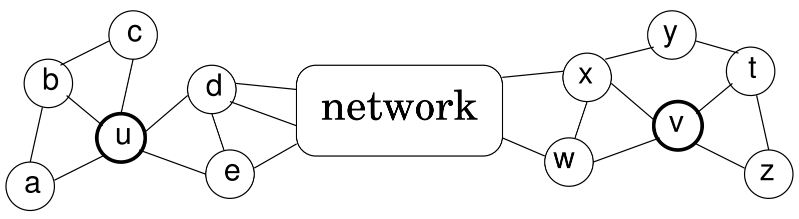


Figure 1.1: An example of two nodes (u and v) that are structurally similar. They have, respectively, degrees 5 and 4, connected to 3 and 2 triangles, and are connected to the rest of the network by two nodes, but are very far apart in the network.

neighborhood (e.g., w steps of a random walk). In a nutshell, nodes that have similar neighborhoods should have similar latent representations. But in all such works, neighborhood is a local concept de ned by some notion of proximity in the network. Thus, two nodes with neighborhoods that are structurally similar but are far apart will not have similar latent representations, which is a fundamental requirement for structural equivalence. Figure 1.1 illustrates the problem, where nodes u and v play similar roles (i.e., have similar local structures) but are very far apart in the network. Since their neighborhoods have no common nodes, recent approaches cannot capture their structural similarity (as we soon show).

It is worth noting why recent approaches for learning node representations such as *DeepWalk* [14] and *node2vec* [12] succeed in some classification tasks but tend to fail in structural equivalence tasks. The key point is that many node features in most real networks exhibit a strong homophily [16] (e.g., two blogs with the same political inclination are much more likely to be connected than at random). Neighbors of nodes with a given feature are more likely to have the same feature. Thus, nodes that are *close* to each other in the network and in the latent representation will tend to share features. Likewise, two nodes that are *far* to each other in the network will tend to be separated in the latent representation, independent of their local structure. Thus, structural equivalence will not properly be captured in the latent representation. However, if classification is performed on features that depend more on structural identity and less on homophily, then such recent approaches are likely to be outperformed by latent representations that better capture structural equivalence.

1.1 Contributions

The main contribution of this work is to provide a flexible framework, called *struc2vec*, for learning latent representations for the structural identity of nodes. This framework offers an alternative and powerful tool to the study of structural identity through the latent space representation. The key ideas within this framework are:

- Assess structural similarity between nodes independently of node and edge attributes, including node labels. Thus, two nodes that are structurally similar will be considered so, independently of their position in the network and node labels in their vicinity. Our approach also does not require the network to be connected, and identifies structurally similar nodes in different connected components.

- Establish a hierarchy to measure structural similarity, allowing progressively more stringent notions of what it means to be structurally similar. In particular, at the bottom of the hierarchy, structural similarity between nodes depend only on their degrees, while at the top of the hierarchy similarity depends on the entire network (from the viewpoint of the node).

- Generates random *contexts* for nodes, which are sequences of structurally similar nodes as observed by a biased random walk (but *not* walking on the original network). Thus, two vertices that frequently appear in similar contexts will likely have similar structure. Such context can be leveraged by language models to learn latent representation for the nodes.

An instance of our framework was implemented and we show its potential through numerical experiments on toy examples and real networks, comparing its performance with *DeepWalk* [14] and *node2vec* [12] – two state-of-the-art techniques for learning latent representations for nodes, and with *RolX* [11] – a recent approach to identify roles of nodes. Our results indicate that while *DeepWalk* and *node2vec* fail to capture the notion of structural identity, *struc2vec* excels on this task – even when the original network is subject to strong random noise (random edge removal). We also show that *struc2vec* is superior in a classification task where node labels depends more on structural identity (i.e., air-traffic networks with labels representing airport activity).

This research gave rise to a paper accepted for publication in the Research Track at the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2017 [17].

1.2 Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents the concept of structural identity of vertices and shows related works that use the network structure to identify or characterize vertices. Chapter 3 is reserved to describe the theoretical concepts that will give a basis to understand the learning of latent representations and to present some models used to learn distributed representations. In Chapter 4, we overview recent related work on learning latent representations of nodes in networks. Chapter 5 presents the *struc2vec*, our proposed framework, in detail. Experimental evaluation and comparison to other methods are shown in Chapter 6. Finally, in Chapter 7 we conclude the text and we point out some promising research directions.

Chapter 2

Structural Identity

This Chapter describes the concept of *Structural Identity* in networks focusing on the different roles exerted by the vertices. We are interested in using solely the network structure to identify vertices, without analyzing node and edge attributes. A literature review will be presented, describing works that use the concepts related to the structural identity of vertices.

2.1 Motivation

A network, in an abstract definition, is a set of actors or objects that have relationships or connections between them. These entities are nodes and the connections between them are edges in the network. We can find numerous networks in our world and many other networks can be modeled based on diverse phenomena [18, 19]. For example, a network created from contacts via e-mail exchanged between people of a company, where the nodes are the people of the company and there is an edge between two people if they have exchanged emails. In a protein-protein interaction (PPI) network, protein molecules (nodes) are in physical contacts (edges) and the network presents molecular associations between chains that occur in a cell [20].

In almost all networks, nodes tend to have one or more functions that greatly determines their role in the network. In a protein-protein interaction (PPI) network proteins exert specific functions based on its structural identity, as catalysis of a cell, for example [3, 4]. In a social network, individuals can hold roles or positions that are defined by social relations [1, 2]. An individual can have the social role 'father' given by social relations within a family, or in a professional context, may have the role 'director', for example. Naturally, an individual with role 'father' will have interactions with other individuals with roles 'mother' and 'child', among others. These roles can be defined by nodes attributes, as gender or age, or by edges attributes, as a relationship. However, from a different point of view, the network structure can be used to define the role or identity of nodes, looking solely for patterns of relations present in the network. Then, each one of these roles is defined by regularities in the patterns of relations between nodes. In this context, we can analyze and identify nodes only by the structure that appears within the network.

Intuitively, different nodes in such networks may perform similar functions, such as digital influencers in a digital social network or managers in a social network of a factory. Although, there are many ways in which two nodes can be similar, nodes can often be partitioned into equivalent classes with respect to their function or role in the network [1]. Nodes in the same equivalent class are structurally similar, that is to say, their relationship to all others are similar. Nodes that are structurally equivalent have similar identity in the network: they share exactly the same structure of neighborhoods around them [5]. For example, in a network of social relations, judges at different courts occupy the equivalent class (or social position) judge, even though they do not work with each other, or even with the same lawyers or attorneys, but they have a pattern of relationships that is structurally similar (with actors that have the same role).

Hubs, cliques, bridges and other structures can be seen as different forms to describing how the nodes in a network are identified on the basis of their patterns of relations with others nodes. For example, nodes in a clique can be very similar structurally because the patterns of ties of these nodes are similar, bridges have a similar function in a way that they connect distinct clusters of the network.

The concept of structural equivalence is often reported with the automorphic equivalence, using the notions of *isomorphism* and *automorphism* [21]. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection function $f : V_1 \to V_2$ that precisely maps the edges of G_1 into edges of G_2, such that f preserves adjacency of vertices, that is, any two nodes u and v of G_1 are adjacent in G_1 if and only if $f(u)$ and $f(v)$ are adjacent in G_2. An isomorphism between G and G itself is called an automorphism of G [22]. Then, we have the following definition:

Definition 1 ([21])**.** Given a graph $G = (V, E)$, two vertices $v_1, v_2 \in V$ are automorphically equivalent if there is an automorphism f of G such that $f(v_1) = v_2$.

Figure 2.1a shows an example of automorphic equivalence: vertices u and v have exactly the same structural identity and exist a automorphism f of G_1 such that $f(u) = v$, that is, if we swap u for v the network remains the same.

However, the problem of determining if two arbitrary graphs are isomorphic belongs to the class NP [23], which means it is possible to computationally check the existence of an isomorphism between two graphs in polynomial time as a function of the size of the graphs, using the vertex mapping as a certificate. But, it is still unknown if this problem can be solved in polynomial time, that is, it is not known

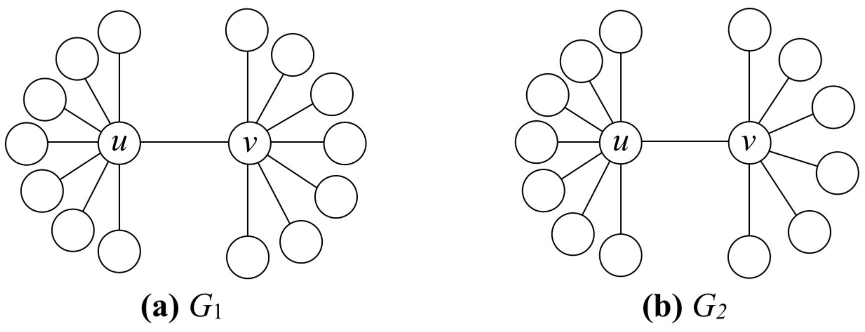


Figure 2.1: In both examples vertices u and v connect their neighbors to the remainder of the graph. (a) Vertices u and u are automorphically equivalent. (b) Vertices u and u are not automorphically equivalent, however they are structural equivalent.

if the problem is in the class P or NP-complete.

Besides that, isomorphism is a binary property. If two nodes are automorphically equivalent and one incident edge to one of these nodes is removed, they will not be automorphically equivalent anymore, even if they have the similar structural identity. This change suddenly makes the equivalence disappear, and thus is a too strong notion of equivalence between vertices. Figure 2.1b shows this issue: intuitively, vertices u and v are structurally similar, they can be seen as small "hubs" who share almost equally the role of connecting the "peripheral" vertices. However they don't have exactly the same structural identity (automorphic equivalence), since u and v have different degrees, so the definition 1 falls short in capturing this notion.

In real-world networks we would like to identify vertices that are structurally similar, even if they are not automorphically equivalent, because exact structural equivalence is likely to be rare, particularly in large networks. It is important to capture the notion that similar vertices, despite not having exactly the same structural identity, play similar roles or functions in the network. This relaxation of the automorphic equivalence is required to properly capture equivalence classes in large real-world networks because we often are interested in analyzing the degree of structural equivalence, rather than the simple presence or absence of exact equivalence.

2.2 Related work

Many different approaches to determine the structural similarity have been proposed in the literature. The most common practices are based on distances or recursions.

Leicht et al. [7] propose a measure of similarity based on the concept that two vertices are similar if their immediate neighbors in the network are themselves similar. This measure can be viewed as a weighted count of the number of paths of all lengths between the vertices in question. They create a self-consistent matrix formulation of similarity that can be evaluated iteratively using the adjacency matrix of the network.

Fouss et al. [8] propose an approach based on a Markov-chain model of random walk through the graph. In a nutshell, they make some calculations using the average commute time and the pseudoinverse of the Laplacian matrix of the graph to provide similarities between any pair of nodes, having the property of increasing when the number of paths connecting those elements increases and when the "length" of paths decreases. The model is evaluated on a collaborative recommendation task where suggestions are made about which movies people should watch based upon what they watched in the past. Experimental results show that the model performs well in comparison with other methods.

Jefferson Simões [24] proposes a definition of local symmetry, based on the structural similarity of neighborhoods around each vertex considering the relationship between the neighborhood of two vertices. Their definition naturally induces a hierarchy of symmetries, which progressively uses more information for classifying vertices, ultimately culminating in the traditional, automorphism-based symmetry, which they call global symmetry.

The celebrated work of Kleinberg [6] uses the network structure of a hyperlinked environment to effective discover and rank pages relevant for a particular topic, defining roles of webpages in the webgraph. They propose and test an algorithmic formulation of the role of authority, based on the relationship between a set of relevant authoritative pages, the most prominent sources of primary content, and the set of role "hub pages", high-quality guides and resource lists, that join them together in the link structure. Hyperlinks encode a considerable amount of latent human judgment, and they use this type of judgment to formulate a notion of authority. Specifically, the creator of page p, by including a link to page q, has in some measure conferred authority on q. They develop a method that, given a query to every web page, assigns to it two scores, called hub score and authority score. They use the link structure to infer a notion of "similarity" among pages using the score generated by the algorithm.

A recent approach to explicitly identify the role of nodes using just the network structure is *RolX* - (Role eXtraction) [11]. This unsupervised approach is based on enumerating various structural features for nodes, finding the more suited basis vector for this joint feature space, and then assigning for every node a distribution over the identified roles (basis), allowing for mixed membership across the roles. Without explicitly considering node similarity or node context (in terms of structure), *RolX* is likely to miss node pairs that are structurally equivalent (to be shown).

Unlike these works, we propose an alternative approach based on unsupervised learning of representations that capture the structural identity of nodes. Let $G = (V, E)$ be a given network, where V are nodes of the network and E its edges $E \subseteq (V \times V)$. Our goal is to learn feature representations $X \in \mathbb{R}^{|V| \times d}$, where d is number of latent dimensions. These latent representations capture the structural identity of nodes in V and obey a relation where embeddings of nodes with exactly or similar roles will be in near positions on the latent space.

Approaches similar to ours have also been recently proposed in the literature, in the sense that it uses representation for nodes in a latent space, such as *node2vec* [12] and *DeepWalk* [14]. However, their goal is not to explicitly capture structural identity of nodes. We discuss these related works in Chapter 4.

Chapter 3

Learning representations

This Chapter presents theoretical background on language models and provides a basis for the understanding of CBOW and Skip-gram, two widely used language models. For both models, initially, the general theoretical concepts are presented, followed by a more in-depth explanation. Finally, computationally efficient approximations, which allow the use of the models in large datasets, will be presented. The sections 3.3.1, 3.3.2 and 3.3.3, for the most part, are based on the work of Xin Rong [25].

3.1 Data representation

The performance of many information processing tasks is heavily influenced by the choice of data representation (or features) on which they are applied. Machine learning algorithms may be more or less efficient when performing a certain task depending on how the input information is represented. For that reason, much effort is put into creating techniques for pre-processing data, transforming it into a meaningful representation that can support more effective machine learning [26].

A feature is a piece of information that might be useful for the task. Feature engineering is a way of using human ingenuity and prior knowledge of the data to create features that make machine learning algorithms perform better. However, this process is very expensive and difficult, and in most times it is not clear how to extract information from the data. To overcome these issues and expand the scope and ease of the use of machine learning, it is important to create learning algorithms that can extract useful information from data that can then be used in classifiers and other predictors.

Feature learning or *representation learning* is a set of techniques that aim to learn features, that is, representations of the data that lead to more effective machine learning tasks. Representation learning has become a field in itself, in the machine learning community, sometimes under the header of Deep Learning [26]. Among the various ways of learning representations, deep learning methods are those that are formed by the composition of multiple non-linear transformations, with the goal of yielding more abstract, and ultimately more useful representations [26].

Natural Language Processing (NLP) is the field of study that focuses on the interactions between human language and computers. Generating dense representations for sparse data has a long history in NLP [27]. Many NLP applications are based on language models that define a probability distribution over sequences of words in a natural language. A fundamental problem that makes language modeling and other learning problems difficult is the *curse of dimensionality*: when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. Traditional encodings, such as one-hot or bag of words, generate representations that have the same size as the vocabulary. This might be an issue since vocabulary size in the order of millions of words is common. The resultant sparse high-dimensional data pose an obstacle for many tasks, including text classification and clustering. To reduce impact of the curse of dimensionality, Bengio et al. [27] proposed a neural network language model that learns distributed representation for words. The model learns simultaneously a distributed representation for each word (called a *word embedding*) along with the probability distribution over word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before receives high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence.

After this work, many neural network language models were developed [26] but none of the previously proposed architectures has been successfully trained on more than a few hundred millions of words, with a modest dimensionality of the word representations [28]. Recently, CBOW and Skip-gram [28, 29] are proposed as architectures to learn high-quality word representations from huge data sets with billions of words, and with millions of words in the vocabulary.

3.2 Language Modeling

The goal of *Language Modeling* is to learn a probability distribution over sequences of words appearing in a language. More formally, let $\mathcal{V}$ denote the set of all words in the language, that is, the vocabulary. A *sentence* in the language is a sequence of words $w_1, w_2, \ldots, w_m$, where $m \geq 1$ and $w_i \in \mathcal{V}$ for $i \in \{1...m\}$ and let $\mathcal{V}^\dagger$ denote the set of all sentences with vocabulary $\mathcal{V}$. According with Michael Collins [30], we have the following definition:

Definition 2 (Language Model)**.** A language model consists of a finite set $\mathcal{V}$, and a function $p(w_1, w_2, \ldots, w_m)$ such that:

1. For any $\langle w_1, w_2, \ldots, w_m \rangle \in \mathcal{V}^\dagger, p(w_1, w_2, \ldots, w_m) \geq 0$

2. In addition,

$$\sum_{\langle w_1, w_2, \ldots, w_m \rangle \in \mathcal{V}^\dagger} p(w_1, w_2, \ldots, w_m) = 1$$

Hence $p(w_1, w_2 \ldots, w_m)$ is a probability distribution over the sentences in $\mathcal{V}^\dagger$.

Given a training corpus (a set of sentences), we would like to learn a function p. Consider a sequence of random variables $X_1, X_2, \ldots, X_m$ where each random variable can take any value in $\mathcal{V}$. We would like to learn the probability of any sequence of words $w_1, w_2, \ldots, w_m$, more precisely, the joint probability:

$$p(X_1 = w_1, X_2 = w_2, \ldots, X_m = w_m) \tag{3.1}$$

Because there are $|\mathcal{V}|^m$ possible sequences of words of the form $w_1, w_2 \ldots, w_m$, it is not feasible for reasonable values of $\mathcal{V}$ and m to list all $|\mathcal{V}|^m$ probabilities. Then, to simplify the model, the following assumption is made:

$$\begin{aligned} & p(X_1 = w_1, X_2 = w_2, \ldots, X_m = w_m) \\ & = p(X_1 = w_1) \prod_{i=2}^{m} p(X_i = w_i \mid X_1 = w_1, \ldots, X_{i-1} = w_{i-1}) && \text{(3.2a)} \\ & = p(X_1 = w_1) \prod_{i=2}^{m} p(X_i = w_i \mid X_{i-1} = w_{i-1}) && \text{(3.2b)} \end{aligned}$$

In the 3.2b step, we have made the assumption that for any $i \in \{2, ..., m\}$, for any $w_1, w_2, \ldots, w_m$:

$$p(X_i = w_i \mid X_1 = w_1, \ldots, X_{i-1} = w_{i-1}) = p(X_i = w_i \mid X_{i-1} = w_{i-1}) \tag{3.3}$$

This is a *Markov assumption*, wich will form the basis of n-gram language models [30]. It was assumed that the occurrence of the i'th word in the sequence depends only on the $i-1$'th word. More specifically, it was assumed that the value of X_i is conditionally independent of $X_1, \ldots X_{i-2}$. This assumption can be generalized in a form that each word only depends on the $n-1$ previous words in the sequence:

$$\begin{aligned} & p(X_i = w_i \mid X_1 = w_1, \ldots, X_{i-1} = w_{i-1}) \\ & = p(X_i = w_i \mid X_{i-(n-1)} = w_{i-(n-1)}, \ldots, X_{i-1} = w_{i-1}) \end{aligned} \tag{3.4}$$

A sequence of n words from a given sentence is a n-gram. An n-gram with $n = 1$ is called a unigram. In an n-gram model, the probability $p(w_1, \ldots, w_m)$ of observing the sentence $w_1, \ldots, w_m$ is given as:

$$p(w_1, \ldots, w_m) = \prod_{i=1}^{m} p(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) \tag{3.5}$$

Language modeling is useful in many natural language processing applications like speech recognition, machine translation, part-of-speech tagging, syntactic parsing, sentiment analysis and information retrieval [26]. Among many ways to learn a language model from a training corpus, a class of models, called *Neural Language Models*, uses neural networks to learn distributed representations of words.

3.3 Neural Language Models

Recent works have focused on using neural networks models to build general representations of words [29] [27]. A *Neural Language Model* is a language model based on neural networks, exploiting their ability to learn distributed representations to reduce the impact of the curse of dimensionality [27]. The neural network learns to associate each word in the vocabulary $\mathcal{V}$ with a latent representation in a continuous vector space with a relatively small number of dimensions. Dimensions of that vector space correspond to a semantic or syntactic structure of human language. The idea is that representations of semantic similar words are closer to each other in space, at least along some dimensions.

In these neural networks, two words can get similar representations if they have semantic and syntactic similarity, that is, if they are functionally similar. This occurs because these words can appear in the same context, helping the neural network to represent compactly a function that makes good predictions on the set of word sequences used to train the model (training set).

Neural Language Models are probabilistic classifiers that learn to predict a probability distribution $P(w_t|\text{context})$, $\forall t \in \mathcal{V}$. So during the training phase, these neural networks learn vectors that are word representations, maximizing a loss function which takes into account the probability distribution of the word sequences. This is done using standard neural network training algorithms such as stochastic gradient descent with back-propagation [27]. The context might be a fixed-size window of previous words (like n-gram), so that the neural network predicts $p(w_i|w_{i-c}, \ldots, w_{i-1})$ from distributed representations of the previous c words [27]. Another option is to use distributed representations of "future" words as well as "past" words as context, so that the estimated probability is $P(w_i|w_{i-c}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+c})$ [29].

The sparsity of the data is a major problem in building language models because it is harder to generalize the statistical learning. When the number of words increases, the number of required sequence examples grows exponentially, but most possible word sequences will not be observed in training.

The advantage of using distributed representations is that it allows the model to generalize well to sequences that are not in the training set of word sequences, but that have words that are similar (syntactically and semantically) to other that were used in sequences present in the training set. Consequently, these words will have similar representations. Since neural networks tend to map nearby inputs to nearby outputs, the predictions corresponding to word sequences with similar representations are mapped to similar predictions. Because many different combinations of features are possible, a very large set of possible meanings can be represented compactly, allowing a model with a comparatively small number of parameters to fit a large training set [27].

Two models to be presented below (CBOW and Skip-gram [29]) aim to learn word embeddings (representations) of words in sequences. Theses models use a large amount of text to create representations of words capturing relationships between them. Such representation capture many linguistic regularities. Training such a lexical model to maximize likelihood will induce word representations with impressive syntactic and semantic properties. For example, it yields a vector approximating the representation for **vec('Rome')** as a result of the vector operation **vec('Paris') – vec('France') + vec('Italy')**.

3.3.1 Continuous Bag-of-Words Model (CBOW)

Continuous Bag-of-Words Model (CBOW) is a Feedforward Neural Network introduced in Mikolov et al. [29]. CBOW is a neural network composed of three layers: an input layer, a single hidden layer, and an output layer. It is trained to predict the target word (e.g. 'eating') from the contextual words that surround it (e.g 'The man is ... in the kitchen.'). More specifically, the goal is to maximize $p(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$ over the training set, where w_t is the input word, $w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}$ are the words in the context and c is the size of the context window. CBOW uses a relaxation of n-gram models where the order of words in a context does not matter. Moreover, CBOW also uses as part of the context the "future", that is, words that appear after w_t.

Figure 3.1 shows the CBOW model. In the input layer, there are C one-hot encoded vectors of size $|\mathcal{V}|$, where C is the number of words in the input context. Each one-hot encoded vector is used to a word at the context. The weights between the input layer and the hidden layer are represented by a $|\mathcal{V}| \times N$ matrix $\boldsymbol{W}$, where N is the size of vector representations of words. The j'th row of $\boldsymbol{W}$ is the N-dimension vector representation $\boldsymbol{v}_{w_j}$ of the word w_j of vocabulary $\mathcal{V}$.

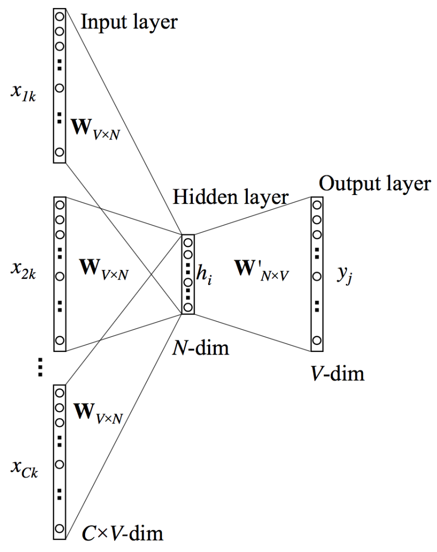


Figure 3.1: Continuous Bag-of-Words Model architecture. Figure from [25].

To compute the output of the hidden layer, CBOW takes the average of the vector representations of the context words. It does this calculating the product of the matrix $\boldsymbol{W}$ by sum of the one-hot vectors of the context words and dividing it by C:

$$\boldsymbol{h} = \frac{1}{C}\boldsymbol{W}^{\mathsf{T}}(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \cdots + \boldsymbol{x}_C) \tag{3.6a}$$

$$= \frac{1}{C}(\boldsymbol{v}_{w_1} + \boldsymbol{v}_{w_2} + \cdots + \boldsymbol{v}_{w_C})^{\mathsf{T}} \tag{3.6b}$$

where $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_C$ are the one-hot encoded vectors of the words $w_1, w_2, \ldots, w_C$ in the context and $\boldsymbol{v}_{w_j}$ (j'th row of $\boldsymbol{W}$) is the vector representation of the word w_j, which is ultimately, the word representation that we want to learn. This implies that the activation function of the hidden layer units is simply linear.

From the hidden layer to the output layer, there is a different weight matrix $\boldsymbol{W'}$ with dimensions $N \times |\mathcal{V}|$. Using these weights, we can compute a score u_j for each word w_j in $\mathcal{V}$:

$$u_j = {\boldsymbol{v'}_{w_j}}^{\mathsf{T}}\boldsymbol{h} \tag{3.7}$$

where $\boldsymbol{v'}_{w_j}$ is the j'th column of matrix $\boldsymbol{W'}$. After this, *softmax function*[1] is used to obtain the probability distribution of words (multinomial distribution):

$$p(w_j \mid w_1, w_2, \ldots, w_C) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{|\mathcal{V}|}\exp(u_{j'})} \tag{3.8}$$

[1]Softmax function is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range [0, 1] that add up to 1.

where y_j is the output of the j'th unit in the output layer and $w_1, w_2, \ldots, w_C$ are words of the input context.

To train the model it is necessary to define a training objective for one training example. CBOW aims to maximize the conditional probability of observing the output word w_O given the input context $w_1, w_2, \ldots, w_C$. It uses a loss function that is commonly used along with the softmax function for training a neural network: cross-entropy. The cross-entropy formula is used to minimize the negative log likelihood of observing the output word w_O given the input context $w_1, w_2, \ldots, w_C$. Then, we consider the logarithm of the conditional probability to define the loss function for one training example:

$$\begin{aligned} E &= -\log p(w_O \mid w_1, w_2, \ldots, w_C) && (3.9a) \\ &= -\log \left(\frac{exp(u_{j^*})}{\sum_{j'=1}^{|\mathcal{V}|} exp(u_{j'})} \right) && (3.9b) \\ &= -\left(u_{j^*} - \log \sum_{j'=1}^{|\mathcal{V}|} exp(u_{j'}) \right) && (3.9c) \end{aligned}$$

where j^* is the index of the actual output word in the output layer (w_O). The loss function can be rewritten as:

$$E = -\sum_{j=1}^{|\mathcal{V}|} t_j \log y_j \qquad (3.10)$$

where $t_j = \mathbb{1}(j = j^*)$, i.e., t_j will only be 1 when the j'th unit is the actual output word, otherwise $t_j = 0$.

With this objective function, it is possible to train the model by computing the gradients with respect to the parameters and at each iteration update them via stochastic gradient descent and backpropagation [31].

Training CBOW

Backpropagation [31] is an usual method for training neural networks. It is used in conjunction with an optimization method such as stochastic gradient descent (SGD). SGD is used to update a set of model parameters (weights of a neural network) in an iterative manner to minimize the loss function. In each iteration, only a subset of training samples from the training set is traditionally used to update the parameters.

Backpropagation is composed of two-phases: propagation and weight updates. In the first phase, an input is fed into the network and propagated forward through the network, layer by layer, until it reaches the output layer. Then, the network output is compared to the desired output, using a loss function, and an error value is calculated for each neuron in the output layer. Next, the error values are propagated backward, starting from the output, such that each neuron has an associated error value which roughly represents its contribution to the original output. Then, backpropagation uses these error values to calculate the gradient of the loss function with respect to the weights in the network. In the second phase, stochastic gradient descent uses the calculated gradients to update the weights, in an attempt to minimize the loss function.

Below are the details to update the weights of the network. In order to use the softmax function in neural networks, it is necessary to compute its derivative. For simplicity, if we define $\sum_{|\mathcal{V}|} = \sum_{j'=1}^{|\mathcal{V}|} exp(u_{j'})$, then the derivative $\frac{\partial y_i}{\partial u_j}$ of the output y_i of the softmax function with respect to its input u_j can be calculated as:

$$\begin{aligned}
\text{if } i = j: \; \frac{\partial y_i}{\partial u_i} &= \frac{\partial \frac{exp(u_i)}{\sum_{|\mathcal{V}|}}}{\partial u_i} = \frac{exp(u_i)\sum_{|\mathcal{V}|} - exp(u_i)exp(u_i)}{(\sum_{|\mathcal{V}|})^2} \\
&= \frac{exp(u_i)}{\sum_{|\mathcal{V}|}} \frac{\sum_{|\mathcal{V}|} - exp(u_i)}{\sum_{|\mathcal{V}|}} = \frac{exp(u_i)}{\sum_{|\mathcal{V}|}}\left(1 - \frac{exp(u_i)}{\sum_{|\mathcal{V}|}}\right) \\
&= y_i(1 - y_i) \\
\text{if } i \neq j: \; \frac{\partial y_i}{\partial u_j} &= \frac{\partial \frac{exp(u_i)}{\sum_{|\mathcal{V}|}}}{\partial u_j} = \frac{0 \, - \, exp(u_i)exp(u_j)}{(\sum_{|\mathcal{V}|})^2} = -\frac{exp(u_i)}{\sum_{|\mathcal{V}|}}\frac{exp(u_j)}{\sum_{|\mathcal{V}|}} \\
&= -\, y_i \, y_j
\end{aligned} \tag{3.11}$$

In order to derive the update equations for the weights of the neural network, it is necessary to use the training objective E defined in (3.10). First of all, the weight update equation between hidden and output layers is achieved taking the derivative of E with respect to j'th unit u_j:

$$\begin{aligned}
\frac{\partial E}{\partial u_j} &= -\sum_{i=1}^{|\mathcal{V}|} \frac{\partial \, t_i \, \log y_i}{\partial u_j} = -\sum_{i=1}^{|\mathcal{V}|} t_i \frac{\partial \, \log y_i}{\partial u_j} = -\sum_{i=1}^{|\mathcal{V}|} t_i \frac{1}{y_i}\frac{\partial y_i}{\partial u_j} \\
&= -\frac{t_j}{y_j}\frac{\partial y_j}{\partial u_j} - \sum_{\substack{i=1 \\ i\neq j}}^{|\mathcal{V}|} \frac{t_i}{y_i}\frac{\partial y_i}{\partial u_j} = -\frac{t_j}{y_j} y_j(1 - y_j) - \sum_{\substack{i=1 \\ i\neq j}}^{|\mathcal{V}|} \frac{t_i}{y_i}(-y_i \, y_j) \\
&= -\, t_j + t_j \, y_j + \sum_{\substack{i=1 \\ i\neq j}}^{|\mathcal{V}|} t_i \, y_j = -\, t_j + \sum_{i=1}^{|\mathcal{V}|} t_i \, y_j = -\, t_j + y_j \sum_{i=1}^{|\mathcal{V}|} t_i \\
\frac{\partial E}{\partial u_j} &= y_j - t_j
\end{aligned} \tag{3.12}$$

Note that the derivative of softmax function calculated in (3.11) was used. Next, we take the derivative of E with respect to w'_{ij} to obtain the gradient on the hidden to output weights:

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial w'_{ij}} = (y_j - t_j)\,\frac{\partial\, \boldsymbol{v'}^{\mathsf{T}}_{w_j}\boldsymbol{h}}{\partial w'_{ij}} = (y_j - t_j)h_i \tag{3.13}$$

Then, using stochastic gradient descent, the weight updating equation for hidden to output weights ($\boldsymbol{W'}$) is obtained:

$$w'^{(new)}_{ij} = w'^{(old)}_{ij} - \eta(y_j - t_j)h_i \tag{3.14}$$

where $\eta > 0$ is the learning rate and h_i is the i'th unit in the hidden layer. Note that it is necessary to apply this update equation for every element of the hidden to output matrix $\boldsymbol{W'}$. Now, we can obtain update equations for input to hidden weights ($\boldsymbol{W}$) using the update equation calculated for $\boldsymbol{W'}$. To do this, first it is taken the derivative of E on the output of the hidden layer:

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{|\mathcal{V}|}\frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{|\mathcal{V}|}(y_j - t_j)\,\frac{\partial\, \boldsymbol{v'}^{\mathsf{T}}_{w_j}\boldsymbol{h}}{\partial h_i} = \sum_{j=1}^{|\mathcal{V}|}(y_j - t_j)\,w'_{ij} \tag{3.15}$$

where u_j is defined in (3.7): input of the j'th unit in the output layer. Now, we take the derivative of E with respect to each element of $\boldsymbol{W}$:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i}\frac{\partial h_i}{\partial w_{ki}} = \sum_{j=1}^{|\mathcal{V}|}(y_j - t_j)\,w'_{ij}\;x_k \tag{3.16}$$

Then, using stochastic gradient descent, the weight updating equation for input to hidden weights ($\boldsymbol{W}$) is obtained:

$$w^{(new)}_{ki} = w^{(old)}_{ki} - \eta\,\frac{1}{C}\,\sum_{j=1}^{|\mathcal{V}|}(y_j - t_j)\,w'_{ij}\;x_k \tag{3.17}$$

where C is the number of words in the input context. It is worth mentioning that the only rows to be updated in $\boldsymbol{W}$ are rows corresponding to input words, because they are the only rows of $\boldsymbol{W}$ whose derivative are non-zero. All the other rows will remain unchanged during this iteration because their derivatives are zero.

3.3.2 Continuous Skip-gram Model

The Skip-gram model is another Feedforward Neural Network introduced in Mikolov et al. [29]. This model is different from the models seen before: it reverses the use of target and context words. Now, the input is only a word and the context words, which are within a certain range before and after the current word, are on the

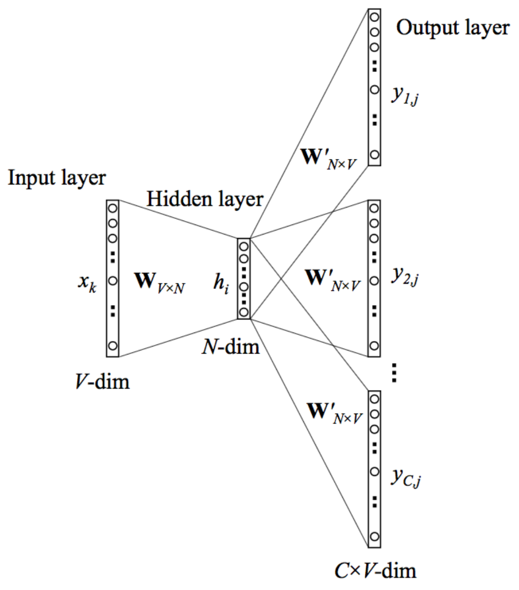


Figure 3.2: Skip-gram Model architecture. Figure from [25].

output layer. Skip-gram is trained to predict a word's context (e.g 'The man is ... in the kitchen.') that surround a word (e.g. 'eating'). More specifically, it aims to maximize $p(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c} \mid w_t)$ over all training corpus, where w_t is the input word, $w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}$ are the words of context and c is the size of the context window. Increasing the size of the context improves the quality of the resulting word representations, but it also increases the computational complexity [29].

Figure 3.2 shows the Skip-gram model. In the input layer, there is only a single one-hot encoded vector of size $|\mathcal{V}|$, corresponding to the input word. The weights between the input layer and the hidden layer are represented by a $|\mathcal{V}| \times N$ matrix $\boldsymbol{W}$, where N is the size of vector representations of words. As the CBOW, the j'th row of $\boldsymbol{W}$ is the N-dimension vector representation $\boldsymbol{v}_{w_j}$ of the word w_j of vocabulary $\mathcal{V}$.

Skip-gram computes the hidden layer output multiplying the matrix $\boldsymbol{W}$ by the input vector:

$$\boldsymbol{h} = \boldsymbol{W}^{\mathsf{T}}\boldsymbol{x} = \boldsymbol{v}_{w_I}^{\mathsf{T}} \tag{3.18}$$

where $\boldsymbol{x}$ is the one-hot encoded vector of the input word w_I and $\boldsymbol{v}_{w_I}$ is the vector representation of the input word w_I. As in CBOW, from the hidden layer to the output layer, there is a different weight matrix $\boldsymbol{W'}$ with dimensions $N \times |\mathcal{V}|$. Using these weights, we can compute a score u_j for each word w_j in $\mathcal{V}$:

$$u_j = \boldsymbol{v'}_{w_j}{}^{\mathsf{T}}\boldsymbol{h} \tag{3.19}$$

where $\boldsymbol{v'}_{w_j}$ is the j'th column of matrix $\boldsymbol{W'}$.

In the output layer, instead of using one softmax as in CBOW, it is necessary to output C independent softmax functions, where C is the number of words in the output context (see Figure 3.2). Each output is computed using the same weight matrix $\boldsymbol{W'}$ with dimensions $N \times |\mathcal{V}|$:

$$p(w_{c,j} = w_{O,c} \mid w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{|\mathcal{V}|} exp(u_{j'})} \tag{3.20}$$

where $w_{c,j}$ is the j'th word on the c'th softmax; $w_{c,O}$ is the c'th word in the context words; w_I is the input word; $u_{c,j}$ and $y_{c,j}$ are the input and output, respectively, of the j'th unit of the c'th softmax. Because the output layer has only one weight matrix ($\boldsymbol{W'}$), we have:

$$u_{c,j} = u_j \quad , \quad c = 1, 2, \dots, C. \tag{3.21}$$

Skip-gram model aims to maximize the conditional probability of observing the context words $w_{O,1}, w_{O,2}, \dots, w_{O,C}$ given the input word w_I. This is achieved by minimizing the negative log likelihood of observing the output context words given the input word w_I. Thus, we consider the logarithm of the conditional probability, assuming conditioned independence, and use it to define the loss function for one training example:

$$E = -\ \log p(w_{O,1}, w_{O,2}, \dots, w_{O,C} \mid w_I) \tag{3.22a}$$

$$= -\ \log \prod_{c=1}^{C} p(w_{O,c} \mid w_I) \tag{3.22b}$$

$$= -\ \log \prod_{c=1}^{C} \frac{exp(u_c)}{\sum_{j'=1}^{|\mathcal{V}|} exp(u_{j'})} \tag{3.22c}$$

$$= -\left(\sum_{c=1}^{C} u_c - C \log \sum_{j'=1}^{|\mathcal{V}|} exp(u_{j'}) \right) \tag{3.22d}$$

where u_c is the score for the c'th word of output context.

With this objective function, we can compute the gradients with respect to the unknown parameters and at each iteration update them via Stochastic Gradient Descent and back-propagation.

Training Skip-gram

Skip-gram is trained using backpropagation in conjunction with stochastic gradient descent (SGD), in a similar fashion to CBOW.

In order to derive the update equations to the weights of neural network, it is necessary to use the training objective E defined at (3.22d). First of all, we derive the update equation of the weights between hidden and output layers taking the derivative of E with respect of each j'th unit on every panel c: $u_{c,j}$:

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} \tag{3.23}$$

which is the prediction error on the layer, the same as in (3.12). Next we take the derivative of E with respect to w'_{ij} to obtain the gradient on the hidden to output weights ($\boldsymbol{W'}$):

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \frac{\partial u_{c,j}}{\partial w'_{ij}} = \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) h_i \tag{3.24}$$

Note that was performed the sum of prediction errors over all context words. Now, using stochastic gradient descent, the weight updating equation for hidden to output weights ($\boldsymbol{W'}$) is obtained:

$$w'^{(new)}_{ij} = w'^{(old)}_{ij} - \eta \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) h_i \tag{3.25}$$

Then, we can obtain update equations for input to hidden weights ($\boldsymbol{W}$) using the update equation calculated for $u_{c,j}$. First, we take the derivative of E on the output of the hidden layer:

$$\begin{aligned}\frac{\partial E}{\partial h_i} &= \sum_{j=1}^{|\mathcal{V}|} \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \frac{\partial u_{c,j}}{\partial h_i} = \sum_{j=1}^{|\mathcal{V}|} \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \frac{\partial \boldsymbol{v'}^{\mathsf{T}}_{w_j} \boldsymbol{h}}{\partial h_i} \\ &= \sum_{j=1}^{|\mathcal{V}|} \sum_{c=1}^{C} (y_{c,j} - t_{c,j})\, w'_{ij}\end{aligned} \tag{3.26}$$

Now, the derivative of E with respect to each element of $\boldsymbol{W}$ is taken:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \frac{\partial h_i}{\partial w_{ki}} = \sum_{j=1}^{|\mathcal{V}|} \sum_{c=1}^{C} (y_{c,j} - t_{c,j})\, w'_{ij}\ x_k \tag{3.27}$$

Finally, using stochastic gradient descent, the weight updating equation for input to hidden weights ($\boldsymbol{W}$) is obtained:

$$w^{(new)}_{ki} = w^{(old)}_{ki} - \eta \sum_{j=1}^{|\mathcal{V}|} \sum_{c=1}^{C} (y_{c,j} - t_{c,j})\, w'_{ij}\ x_k \tag{3.28}$$

The understanding of (3.28) is similar as that for (3.17).

3.3.3 Optimizing Computational Efficiency

The models seen before (CBOW and Skip-gram) usually are huge neural networks, that is, they have big weight matrices because the size of vocabulary $\mathcal{V}$ is huge (10^5 – 10^7 terms) [29]. For these models, there are two vector representations for each word in the vocabulary: the input vector $\boldsymbol{v}_w$, and the output vector $\boldsymbol{v'}_w$. Learning the input vectors is cheap but learning the output vectors is very expensive. In order to update $\boldsymbol{v'}_w$, for each training instance (or a mini-batch), it is necessary to iterate through every word w_j in the vocabulary, compute its score u_j and its predicted probability y_j, calculate the derivatives using the back-propagation, and, finally, update the output vector $\boldsymbol{v'}_{w_j}$. Then, as can be seen in equations (3.9c) and (3.22d), for a given training sample it is necessary to evaluate/update $O(|\mathcal{V}|)$ network units. This formulation is impractical because doing such computations for all words, for every training instance is very expensive. A good approach to overcome this is to approximate the softmax function.

Hierarchical Softmax

A computationally efficient approximation of the full softmax is the hierarquical softmax [32]. The main advantage is that instead of evaluating $O(|\mathcal{V}|)$ output nodes in the neural network to obtain the probability distribution, we only need to evaluate $O(\log |\mathcal{V}|)$ nodes.

The hierarchical softmax uses a binary tree representation of the output layer with the $|\mathcal{V}|$ words as its leaves. For each leaf unit, there exists a unique path from the root to the unit. Each node on the path is responsible for making a decision about which of its children it should go to. This is done by learning a binary classifier that chooses a child node given an input. The total probability of a word is given by the product of the probabilities of the correct decision of each binary classifier at each node on the path from the root to the leaf node corresponding to that word. Figure 3.3 shows an example tree.

In the hierarchical softmax there is no output vector representation ($\boldsymbol{v'}_w$) for words. Instead, each of $|\mathcal{V}| - 1$ units has a output vector $\boldsymbol{v'}_{n(w,j)}$. Let $n(w,j)$ denote the j'th node on the path from the root to w, and let $L(w)$ denote the length of this path, so $n(w,1) = \mathit{root}$ and $n(w, L(w)) = w$. Then the hierarchical softmax defines the probability of a word being the output word as:

$$p(w = w_O) = \prod_{j=1}^{L(w)-1} \sigma\Big([\![n(w,j+1) = \mathrm{ch}(n(w,j))]\!] \cdot \boldsymbol{v'}^{\mathsf{T}}_{n(w,j)} \boldsymbol{h} \Big) \tag{3.29}$$

where $\sigma(x) = 1/(1 + exp(-x))$, the sigmoid function; $\mathrm{ch}(n)$ is the left child of unit n; $[\![x]\!]$ is 1 if x is true and -1 otherwise; $\boldsymbol{v'}_{n(w,j)}$ is the output vector of the inner

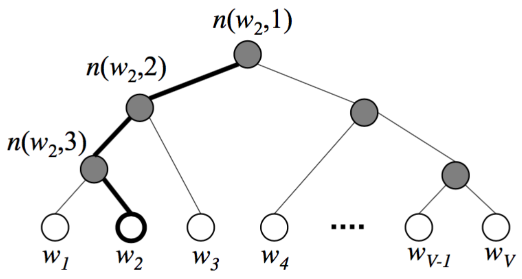


Figure 3.3: An example binary tree for the hierarchical softmax model. The white units are words in the vocabulary, and the dark units are inner units. An example path from root to w_2 is highlighted. In the example shown, the length of the path $L(w_2) = 4$. $n(w, j)$ means the j'th unit on the path from root to the word w. Figure from [25].

unit $n(w, j)$; $\boldsymbol{h}$ is the output value of the hidden layer in the CBOW and skip-gram models. Since a balanced binary tree has a depth of $O(\log(|\mathcal{V}|))$, it is only necessary to evaluate $O(\log(|\mathcal{V}|))$ nodes to obtain the final probability of a word.

In contrast to the CBOW or Skip-gram with regular softmax formulation, which assigns two vector representations $\boldsymbol{v}_{w_j}$ and $\boldsymbol{v}'_{w_j}$ to each word w_j, hierarchical softmax formulation has one vector representation $\boldsymbol{v}_{w_j}$ for each word w_j and one vector representation $\boldsymbol{v}'_n$ for every inner node n of the binary tree.

Now, the loss function for one example is defined as:

$$E = -\log p(w = w_O \mid w_I) \tag{3.30a}$$

$$= -\sum_{j=1}^{L(w)-1} \log \sigma\Big([\![n(w, j+1) = \mathrm{ch}(n(w,j))]\!] \cdot {\boldsymbol{v}'}^{\top}_{n(w,j)} \boldsymbol{h} \Big) \tag{3.30b}$$

This loss function can be used for both CBOW and skip-gram models. When used for skip-gram model, we need to repeat this update procedure for each of the C words in the output context.

The tree used by the hierarchical softmax has a considerable effect on the performance. The computational complexity per training instance is reduced from $O(|\mathcal{V}|)$ to $O(\log |\mathcal{V}|)$, which is a big improvement in speed. However, computing the probability of all $|\mathcal{V}|$ words will remain expensive even with the hierarchical softmax. Moreover, the model has roughly the same number of parameters, more more specifically $|\mathcal{V}| - 1$ vectors for inner-units compared to originally $|\mathcal{V}|$ vector representations for words.

Negative Sampling

Negative samping (NEG) [29] is another efficient way to perform the computation of the updates of word output vectors. NEG is a simplification of Noise Contrastive

Estimation (NCE) [33], and can be shown that NCE approximately maximizes the log probability of the softmax.

In order to deal with the difficulty of having too many output vectors that need to be updated per each training example, NEG only updates a small percentage of them. NEG also uses a logistic loss function to minimize the negative log-likelihood of words in the training set.

The idea is that the output word (i.e., the ground truth, or positive sample) should be kept in the sample and gets its representation updated, and it is necessary to sample a few words as negative samples (hence "negative sampling"). A probabilistic distribution is needed for the sampling process. This distribution is called the noise distribution and is denoted as $P_n(w)$. One can determine a good distribution empirically[2].

Mikolov et al. [29] defines NEG by the loss function:

$$\begin{aligned} E &= -\log p(w = w_O \mid w_I) && \text{(3.31a)} \\ &= -\log \sigma(\boldsymbol{v'}^{\mathsf{T}}_{w_O} \boldsymbol{h}) - \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log \sigma(-\boldsymbol{v'}^{\mathsf{T}}_{w_j} \boldsymbol{h}) && \text{(3.31b)} \end{aligned}$$

where w_O is the output word (i.e., the positive sample), and $\boldsymbol{v'}_{w_O}$ is its output vector; $\boldsymbol{h}$ is the output value of the hidden layer: $\boldsymbol{h} = \frac{1}{C}\sum_{c=1}^{C} \boldsymbol{v}_{w_c}$ in the CBOW model and $\boldsymbol{h} = \boldsymbol{v}_{w_I}$ in the Skip-gram model; $\mathcal{W}_{\text{neg}} = \{w_j \mid j = 1, \ldots, K\}$ is the set of words (K words) that are sampled based on $P_n(w)$, i.e., negative samples. This loss function can be used for both CBOW and the Skip-gram model. For the Skip-gram model, it is necessary to apply the update process for one context word at a time.

Then, the update process only needs to be applied to $w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}$ instead of every word in the vocabulary. This saves a significant amount of computational effort time per training example.

[2]Mikolov et al. [29] investigated a number of choices for $P_n(w)$ and suggest using a unigram distribution raised to the 3/4'th power, for the best quality of results.

Chapter 4

Learning representations from networks

In this Chapter, we present recent works that brought the representation learning for the context of networks. More specifically, *DeepWalk* and *node2vec* are two frameworks for learning latent representations for nodes. Finally, a comparison between the methods is presented showing their main differences and performances in classification tasks.

4.1 Introduction

Many important tasks in network analysis can require predictions over objects (nodes) or its relationships (edges). In a typical network classification task, we are interested in predicting the labels of nodes [12, 14]. For example, in a social network, we might be interested in predicting interest groups of users, or in a protein-protein interaction network we might be interested in predicting functions of proteins [3, 4]. Furthermore, in link prediction, we yearn to predict if a pair of vertices in a network will have an edge connecting them.

In machine learning, many methods employ as input informative, discriminating and independent features. This means that to use networks in machine learning tasks it is necessary to construct a feature vector representation for the nodes or edges. A typical solution involves hand-engineering domain-specific features based on intuition and knowledge of the domain experts. However, usually features are designed for specific tasks and can not generalize well across different prediction tasks. As shown in Chapter 3, an alternative approach is to learn feature representations by solving an optimization problem.

As we described in Chapter 3, Skip-gram [28, 29] was proposed as a technique to learn dense representations for text data, providing an easy optimization problem where a word's context should be predicted given its latent representations. Moreover, the embeddings capture word meanings, placing semantically similar words near each other in the latent space.

Due to the high-dimensional and often sparse nature of graph representations (e.g. the adjacency matrix), learning node embeddings is equally important for machine learning applications on network data. Since Skip-gram (and most other language models) requires temporal sequences as input, adapting it to learn representations for graphs is non-trivial as graph data is not linear.

Learning a language model from a network was first proposed by *DeepWalk* [14]. It proposes to use sequences of nodes from a graph, which are then treated as sentences by Skip-gram. In a text, due to linearity, the notion of a neighborhood can be simply defined using a sliding window over consecutive words. However, networks are not linear, and thus a proper notion of a neighborhood is needed. A reasonable and cheap way to define a neighborhood of a vertex is using sequences of vertices taken from random walks. Intuitively, since vertices in the same Skip-gram window are close in the network, the learned representations capture mostly vertices that are close in the network.

The idea was later extended by *node2vec* [12]. By proposing a biased 2nd order random walk model, it provides more flexibility when generating the context of a vertex. In its framework, biased random walks are designed to capture both vertex proximity and structural equivalence.

subgraph2vec [13] is another recent approach that aims to learn latent features for rooted subgraphs, and unlike the previous techniques it does not use random walks to generate context for nodes. Alternatively, it proposes Radial Skip-gram, a modification of the original Skip-gram where the context of a node is simply defined by its neighbors. Additionally, *subgraph2vec* properly captures structural equivalence by anchoring equivalent vertices to the same point in the latent space. Nonetheless, the notion of structural equivalence is very rigid since it is defined as a binary property dictated by the Weisfeiler-Lehman isomorphism test [34].

All these works have used neural language models for learning latent representations of vertices in a network. They take a graph as input and produces a latent representation of vertices as an output. More specifically, let $G = (V, E)$ denote the network under consideration with vertex set V and edge set E, where $n = |V|$ denotes the number of nodes in the network. The goal is to learn features $X \in \mathbb{R}^{|V| \times d}$, where d is number of dimensions of the latent representations. Instead of using words and sequences of words to create word representations as traditional language models, these works use vertices and sequences of vertices generated by random walks to create representations of vertices. The idea is that nodes that have similar neighborhoods or roles in the network should have similar latent representations.

These latent representations can be easily used for machine learning tasks, such as node classification.

4.2 *DeepWalk*

DeepWalk [14] was a pioneering work that brought representational learning to networks. *DeepWalk* generalized advancements in language modeling (Skip-gram) from sequences of words to graphs. It uses local information obtained from truncated random walks to learn latent representations of nodes that encode social relations in a continuous vector space, which can be exploited by machine learning algorithms.

The result of applying *DeepWalk* to the well-studied Karate network [35] is shown in Figure 4.1. The graph, presented by force-directed layout, is shown in Figure 4.1a. Figure 4.1b shows the output of the method with 2 latent dimensions. Beyond the striking similarity, it is possible to note that linearly separable portions of (4.1b) correspond to clusters found through modularity maximization in the input graph (4.1a), shown as vertex colors.

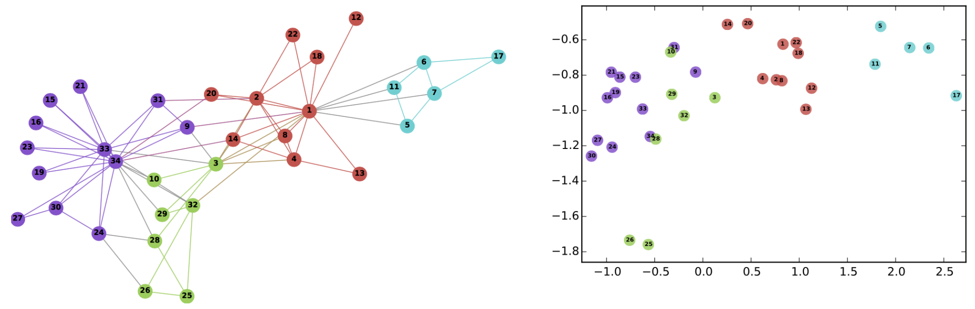

(a) Input: Karate Graph (b) Output: Representation

Figure 4.1: DeepWalk is used on Zachary's Karate network [35] to generate node representations in $\mathbb{R}^2$. Figure from [14].

The key idea behind this work was treating sequences of vertices, generated by random walks, as sentences. If the degree distribution of a network follows a power law[1], we observe that the frequency which vertices appear in the random walks will also follow a power-law distribution. The authors argue that a similar phenomenon appears in context of natural language: word frequency follows a similar power-law distribution. Thus, techniques like Skip-gram, which have been used to model natural language, where the symbol frequency follows a distribution similar to a power law, can be applied to model community structure in networks.

[1]A scale-free network is a network whose degree distribution follows a power law, at least asymptotically. That is, the fraction $P(k)$ of nodes in the network having k connections to other nodes goes for large values of k as $P(k) \sim k^{-\gamma}$, where γ is a parameter whose value is typically in the range $2 < \gamma < 3$ [19].

As in any language modeling algorithm, the required input is a corpus (sentences) and a vocabulary $\mathcal{V}$. *DeepWalk* considers a set of sequences of vertices, generated by short truncated random walks, as a corpus, and the graph vertices as vocabulary ($\mathcal{V} = V$). The process of generating random walk takes a graph G and samples uniformly a random vertex v_i as the root of the random walk $\mathcal{W}_{v_i}$. A walk samples uniformly from the neighbors of the last vertex visited until the maximum length (t) is reached.

DeepWalk specifies the number of random walks γ of length t to start at each vertex. After generating all the vertex sequences using random walks, these sequences are used by Skip-gram to generate node representations, through the neural network training process. To train the Skip-gram, it is necessary to define the size of the window (w) to be used as context, that is, given the representation of a node v_j, we would like to maximize the probability of its neighborhood in the walk (inside w).

4.3 *node2vec*

node2vec [12] extends *DeepWalk* defining a flexible notion of a node's network neighborhood. It generalizes *DeepWalk*, which is based solely on random walks to construct neighborhoods, learning representations that embed nodes from the same network community closely together, as well as learning representations for nodes that have similar roles. In particular, nodes in networks could be organized based on communities they belong to (i.e., homophily) and in other cases, the organization could be based on the structural roles (i.e., structural equivalence). *node2vec* uses a biased 2nd order random walk approach to generate (sample) network neighborhoods for nodes.

In this work the problem of sampling node's neighborhood is viewed as a form of local search. Generally, there are two extreme strategies for generating neighborhoods: *Breadth-first Search (BFS)*: The neighborhood is restricted to nodes which are immediate neighbors of the source; and *Depth-first Search (DFS)*: The neighborhood consists of nodes sequentially sampled at increasing distances from the source node. The neighborhoods sampled by BFS lead to representations that correspond closely to structural equivalence. In order to ascertain structural equivalence, it is often sufficient to characterize the local neighborhoods accurately. For example, structural equivalence based on network roles such as bridges and hubs can be inferred just by observing the immediate neighborhoods of each node. In DFS, the sampled nodes more accurately reflect a more global view of the neighborhood which is required to infer communities based on homophily.

Considering the above observations, *node2vec* applies a flexible neighborhood sampling strategy using a biased random walk procedure, which allows smoothly

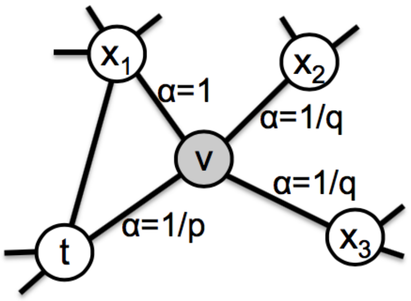


Figure 4.2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α. Figure from [12].

interpolation between BFS and DFS, exploring neighborhoods in a BFS as well as in a DFS fashion.

node2vec has the same parameters as *DeepWalk*: γ random walks of length t to start at each vertex and the size of the window (w) to be used as context by Skip-gram. However, two parameters p and q are used in order to give weights to edges to bias the steps of the random walk. Consider a random walk that traversed edge (t, v) and resides at node v (see Figure 4.2). The random walk needs to decide on the next step so it evaluates the transition probabilities on edges (v, x) leading from v. The unnormalized transition probability is set to $\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$, where:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{4.1}$$

and d_{tx} denotes the shortest path distance between nodes t and x, and must be one of 0, 1, 2. Parameters p and q control how the random walk explores the neighborhood:

- **Return parameter, p**: p controls the likelihood of immediately revisiting a node in the random walk. Setting it to a high value ($> \max(q, 1)$) ensures that the walk is less likely to sample an already-visited node in the following two steps (unless the next node in the walk had no other neighbor). This encourages moderate exploration and avoids 2-hop redundancy in sampling. If p is low ($< \min(q, 1)$), it would lead the walk to backtrack a step, keeping the walk close to the starting node u.

- **In-out parameter, q**: q allows the search to differentiate between "inward" and "outward" nodes. If $q > 1$, the random walk is biased towards nodes close to node t (see Figure 4.2). Such walks obtain a local view of the underlying graph with respect to the start node in the walk and approximate BFS behavior

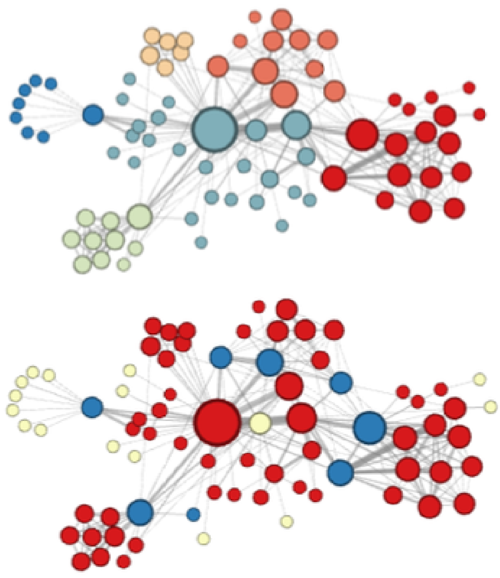


Figure 4.3: Visualizations of Les Misérables network generated by *node2vec* with label colors reflecting homophily (top) and structural equivalence (bottom). Figure from [12].

in the sense that the samples comprise of nodes within a small locality. On the other hand, if $q < 1$, the walk is more inclined to visit nodes which are further away from the node t, encouraging outward exploration (DFS behavior).

To demonstrate the framework and to illustrate the use of its parameters, the network Les Misérables [36] is used as input to *node2vec*. In this network, nodes are characters in the novel Les Misérables and edges correspond to a joint action of two characters. Figure 4.3(top) shows the example created using $p = 1$, $q = 0.5$. Network communities are colored using the same color. In this setting, the algorithm discovers communities of characters that frequently interact with each other in the sub-plots of the novel.

In order to discover which nodes have the same structural roles, the parameters have been set as $p = 1$, $q = 2$. In this experiment *node2vec* obtains a complementary assignment of node to clusters such that the colors correspond to structural equivalence as illustrated in Figure 4.3(bottom). For instance, the algorithm embeds blue-colored nodes close together in the latent space. These nodes represent characters that act as bridges between different sub-plots of the novel.

Lastly, *node2vec* is a semi-supervised algorithm and its parameters can be learned directly using a fraction of labeled data.

4.4 Multi-label classification

One way to compare the quality of representations generated by the aforementioned methods is to evaluate their performance on a multi-label classification problem. In the multi-label classification setting, every node is assigned one or more labels from a finite set L. The node feature representations are input to a one-vs-rest logistic regression classifier with L2 regularization. During the training phase, we observe a certain fraction of nodes and all their labels. The task is to predict the labels for the remaining nodes. Grover and Leskovec [12] use three datasets to perform this experiment:

- BlogCatalog [37]: This is a network of social relationships of the bloggers listed on the BlogCatalog website. The labels represent blogger interests inferred through the metadata provided by the bloggers. The network has 10,312 nodes, 333,983 edges, and 39 different labels.
- Protein-Protein Interactions (PPI) [38]: This is a subgraph of the PPI network for Homo Sapiens. The subgraph corresponds to the graph induced by nodes for which we could obtain labels from the hallmark gene sets [19] and represent biological states. The network has 3,890 nodes, 76,584 edges, and 50 different labels.
- Wikipedia [39]: This is a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. The labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger [32]. The network has 4,777 nodes, 184,812 edges, and 40 different labels.

Besides *DeepWalk* and *node2vec*, the performance of two other methods are also evaluated:

- Sprectral Clustering [40]: This is a matrix factorization approach in which we take the top d eigenvectors of the normalized Laplacian matrix of graph G as the feature vector representations for nodes.
- LINE [41]: This method learns d-dimensional feature representations in two separate phases. In the first phase, it learns $d/2$ dimensions by BFS-style simulations over immediate neighbors of nodes. In the second phase, it learns the next $d/2$ dimensions by sampling nodes strictly at a 2-hop distance from the source nodes.

Figure 4.4 summarizes the results for the Micro-F1 and Macro-F1 scores and also compares performance while varying the train-test split from 10% to 90%.

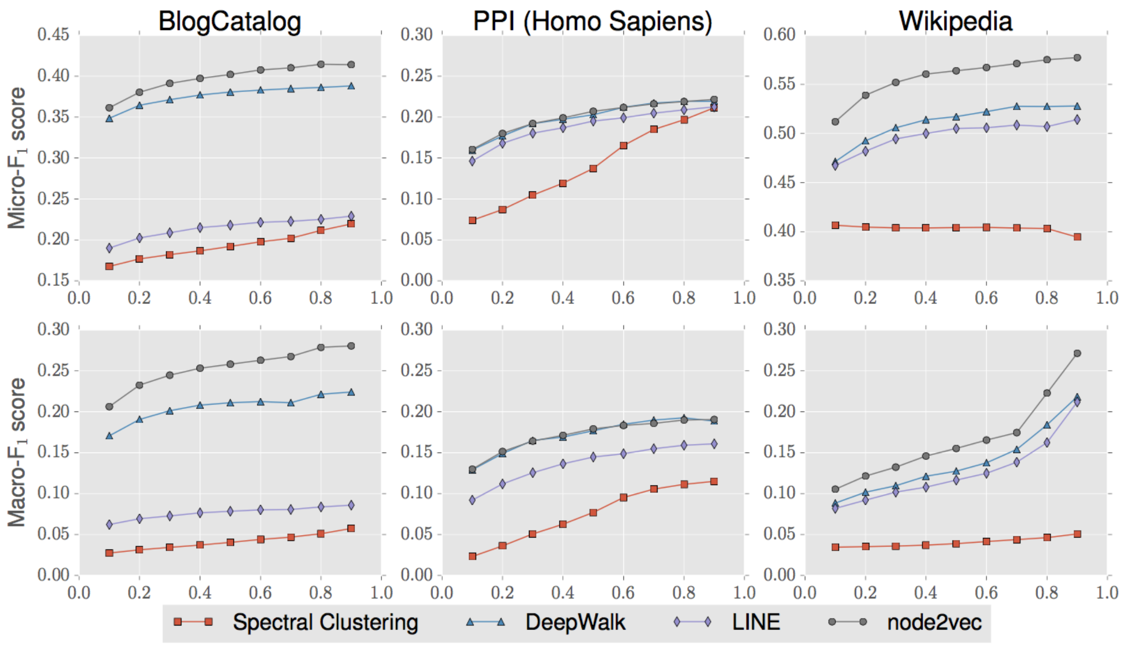


Figure 4.4: Performance evaluation of different benchmarks on varying the amount of labeled data used for training. The x axis denotes the fraction of labeled data, whereas the y axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores, respectively. *DeepWalk* and *node2vec* give comparable performance on all networks. Figure from [12].

Parameters p and q (for *node2vec*) are learned using 10-fold cross-validation on 10% labeled data with a grid search over $p, q \in 0.25, 0.50, 1, 2, 4$. *DeepWalk* and *node2vec* have the best performances. Yet, *node2vec* performs better in almost all experiments with the three networks. These results are due to the ability of the *node2vec* to capture both homophily and structural equivalence from the networks.

However, even though the experiments suggest that *node2vec* can capture structural equivalence, it is unclear how it would perform on larger graphs. More specifically, structurally equivalent vertices will never share the same context if their distance (hop count) is larger than the Skip-gram window w. In the next chapter, we propose a framework that overcomes this limitation, generating node latent representations that do not depend on node neighborhood.

Chapter 5

Framework *struc2vec*

This Chapter presents and describes the framework *struct2vec*, a general methodology for learning latent representations for the structural identity of nodes. First, we propose a measure of structural similarity of nodes, using Dynamic Time Warping (DTW). Next, we create a multilayer weighed graph that encodes structural similarity between nodes. Afterward, is shown how we generate node sequences that will be used as input for Skip-gram. Lastly, we present practical optimizations of the model to decrease the computational complexity.

5.1 Introduction

Consider the problem of learning latent representations for nodes that captures their structural identity in the network. A successful approach should exhibit two desired properties:

- The distance between the latent representation of nodes should be strongly correlated to their structural similarity. Thus, two nodes that are identical from the network structure point of view should have the same latent representation, while nodes with different structural identities should be far apart.
- The latent representation should not depend on any node or edge attribute, including the node labels. Thus, structurally similar nodes should have close latent representation, independent of node and edge attributes in their neighborhood. The structural identity of nodes must be independent of its "position" in the network.

Given these two properties, we propose *struct2vec*, a general methodology for learning latent representations for nodes. The methodology is composed of four main steps, informally defined as follows:

1. **Measure structural similarity:** Determine the structural similarity between each vertex pair in the graph for different neighborhood sizes. This induces a hierarchy in the measure for structural similarity between nodes, providing more information to assess structural similarity at each level of the hierarchy.

2. **Construct the multilayer graph:** Construct a weighted multilayer graph where all nodes in the network are present in every layer, and each layer corresponds to a level of the hierarchy in measuring structural similarity. Moreover, edge weights among every node pair within each layer are inversely proportional to their structural similarity.

3. **Generate context for vertices:** Use the multilayer graph to generate context for each node. In particular, biased random walks on the multilayer graph are used to generate node sequences. These sequences are likely to include nodes that are more structurally similar.

4. **Learn a language model:** Apply a technique to learn latent representation from a context given by the sequence of nodes, for example, Skip-Gram.

Note that *struct2vec* is quite flexible as it does not determine any particular structural similarity measure or representational learning framework. In what follows, we explain in detail each step of *struct2vec* and provide a rigorous approach to a hierarchical measure of structural similarity.

5.2 Measuring structural similarity

The first step of *struct2vec* is to determine the structural similarity of node pairs without using any node or edge attributes. Moreover, we need a measure that can cope with increasing neighborhood sizes. While there are many ways to measure the *structural similarity* between two vertices, we would like a metric with the following property.

Let $G = (V, E)$ denote the network under consideration with vertex set V and edge set E, where $n = |V|$ denotes the number of nodes in the network and k^* its diameter. Let $N_k(u)$ denote the set of nodes with distance less than or equal to $k \geq 0$ from $u \in V$ (note that $N_0(u) = u$ and $N_1(u)$ are the neighbors of u and u itself, see Figure 5.1b). Let $G[S]$ denote the induced subgraph of G over the set of nodes $S \subset V$. Note that $G[N_1(u)]$ is often referred to as the *egonet* of node u.

Let $f(u, v) \geq 0$ denote a distance measure for the structural similarity between $u, v \in V$. A suitable f should satisfy the following property:

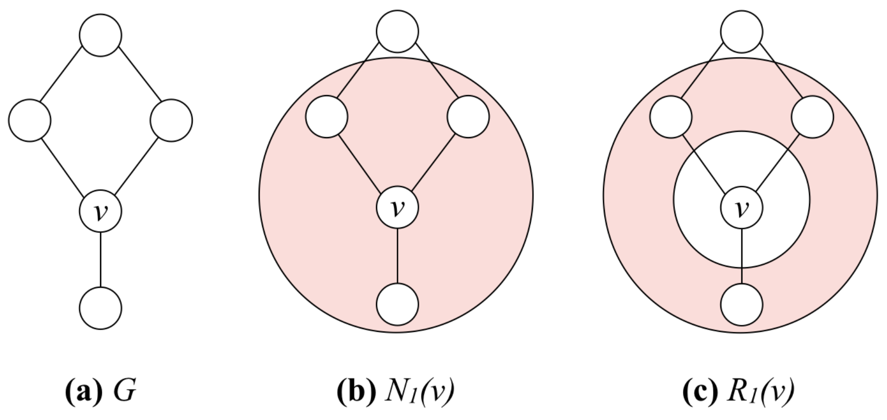


Figure 5.1: (a) An example graph G. (b) $N_1(v)$: set of nodes with distances less than or equal to $k = 1$. (c) $R_1(v)$: set of nodes at distance exactly $k = 1$.

- $f(u, v) = 0$ if there exists an isomorphism between $G[N_k(u)]$ and $G[N_k(v)]$ for any $k > 0$, mapping u onto v.

Under this property, two nodes that have locally isomorphic neighborhoods should be considered identical to one another, and thus, have a structural distance of zero. Clearly, this property is desired when considering the structural identity of nodes in networks.

However, isomorphisms cannot be used directly to measure structural similarity. For one reason, there are no polynomial time algorithm to determine if two arbitrary graphs are isomorphic, and second, isomorphism is a binary property. Thus, we consider the following approach.

Let $s(S)$ denote the ordered degree sequence of the set of vertices $S \subset V$. Note that if $G[N_k(u)]$ is isomorphic to $G[N_k(v)]$ for any $k > 0$, mapping u to v, then $s(N_{k-1}(u)) = s(N_{k-1}(v))$. Namely, the ordered degree sequences of nodes in the $(k-1)$-hop neighborhood of u and v must be identical. Thus, the ordered degree sequence can lead to a distance metric that satisfies the desired property. Moreover, the degree sequence avoids any label information associated to nodes or edges.

The ordered degree sequence is also a natural choice for inducing a hierarchy of distance functions. Let $R_k(u)$ denote the set of nodes at distance exactly $k \geq 0$ from u in G (see Figure 5.1c). Thus, $R_k(u) = N_k(u) \setminus N_{k-1}(u)$ for $k \geq 0$ (let $N_{-1}(u) = \emptyset$). By comparing the ordered degree sequences of the rings of nodes at distance k from both u and v we can impose a hierarchy in assessing their structural similarity, that becomes more stringent as k increases. In particular, let $f_k(u, v)$ denote the *structural distance* between u and v when considering their k-hop neighborhoods (all nodes at distance less than or equal to k and all edges among them). In particular, we define:

$$\begin{aligned} f_k(u,v) \; = & f_{k-1}(u,v) \; + \; g(s(R_k(u)), s(R_k(v))), \\ & k \geq 0 \;\text{ and }\; |R_k(u)|, |R_k(v)| > 0 \end{aligned} \tag{5.1}$$

where $g(D_1, D_2) \geq 0$ measures the distance between the ordered degree sequences D_1 and D_2 and $f_{-1} = 0$. Note that by definition $f_k(u,v)$ is non-decreasing in k and is defined only when both u or v have nodes at distance k. Moreover, using the ring at distance k in the definition of $f_k(u,v)$ forces the comparison between the degree sequences of nodes that are at the same distance from u and v. Finally, note that if $G[N_k(u)]$ and $G[N_k(v)]$ are isomorphic for some $k > 0$, mapping u onto v, then $f_{k-1}(u,v) = 0$.

A final step is determining the function that compares two degree sequences. Note that $s(R_k(u))$ and $s(R_k(v))$ can be of different sizes and its elements are arbitrary integers in the range $[0, n-1]$ with possible repetitions, where $n = |V|$ (i.e., any possible degree). We adopt Dynamic Time Warping (DTW) to measure the distance between two ordered degree sequences, a technique that can cope better with sequences of different sizes and loosely compares sequence patterns [42, 43].

5.2.1 Dynamic Time Warping (DTW)

Dynamic time warping (DTW) distance measure is a well-known technique to find the optimal alignment between two (time-dependent) sequences under certain restrictions [44]. The sequences are "warped" non-linearly in the time dimension to determine a measure of their similarity independent of certain non-linear variations in the time dimension. It is often used to determine time series similarity, classification, and to find corresponding regions between two time series. Originally, DTW has been used to compare different speech patterns in automatic speech recognition [43].

Informally, DTW aims to find the optimal alignment between two sequences X and Y. Given a distance function $\text{dist}(x,y)$ for the elements of the sequence, DTW matches each element $x \in X$ to $y \in Y$, such that the sum of the distances between matched elements is minimized. Note that each element in one sequence can be matched to more than one element in the other, but crossings in the matching are not allowed, and all elements must be matched. Thus, two sequences that are identical except for localized stretching of the time axis will have DTW distances of zero. An example of how one sequence can be "warped" to another is shown in Figure 5.2.

More formally, suppose we have two time-dependent sequences $X = x_1, x_2, \ldots, x_i, \ldots, x_m$ and $Y = y_1, y_2, \ldots, y_j, \ldots, y_n$ of lengths m and n. The goal is

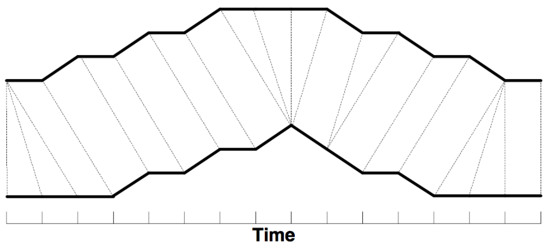


Figure 5.2: A warping between two time-dependent sequences. Figure from [43].

to find an alignment path between X and Y having minimal overall cost. A warp path (or alignment path) W is defined as:

$$W = w_1, w_2, \ldots, w_K \qquad \max(m, n) \leq K < m + n \tag{5.2}$$

where K is the length of the wrap path and the k'th element of the warp path is $w_k = (i, j)$, where i and j are the index from sequences X and Y, respectively. The alignment path must satisfy to the following criteria [45]:

- Boundary condition: $w_1 = (1, 1)$ and $w_K = (m, n)$. The starting and ending points of the warping path must be the first and the last points of aligned sequences.
- Monotonicity: Given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a–a' \geq 0$ and $b - b' \geq 0$. This forces the points in W to be monotonically spaced in time.
- Continuity: Given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a–a' \leq 1$ and $b - b' \leq 1$. This restricts the allowable steps in the warping path to adjacent cells, including diagonally adjacent cells.

The optimal warp path is the minimum-distance warp path, where the distance of a warp path W is:

$$\text{dist}(W) = \sum_{k=1}^{k=K} \text{dist}(w_{ki}, w_{kj}) \tag{5.3}$$

where $\text{dist}(w_{ki}, w_{kj})$ is the distance between the two data points $x \in X$ and $y \in Y$ indexes in the k'th element of the warp path. In order to find the optimal warp path, we need to test every possible warping path between X and Y. It can be computationally challenging due to the exponential growth of the number of optimal paths as the lengths of X and Y grow linearly. To overcome this challenge, a dynamic programming approach is used to find this optimal warp path.

DTW starts by building a $m \times n$ matrix D where the $D(i, j)$ element of the matrix contains the minimum-distance warp path that can be constructed from the two sequences $X' = x_1, \ldots, x_i$ and $Y' = y_1, \ldots, y_j$. The $D(m, n)$ element will contain

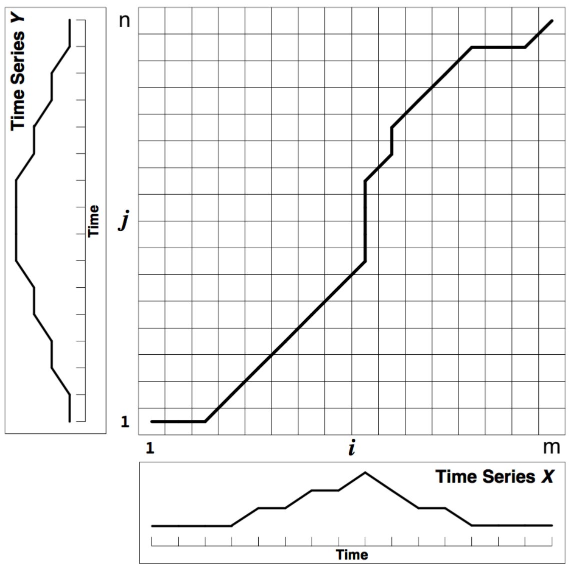


Figure 5.3: A cost matrix with the minimum-distance warp path traced through it. Figure from [43].

the minimum-distance warp path between X and Y. Intuitively, such an optimal path runs along a "valley" of low cost within the matrix D. The x-axis is the time of sequence X, and the y-axis is the time of sequence Y. Figure 5.3 shows an example of a cost matrix and a minimum-distance warp path traced through it from $D(1, 1)$ to $D(m, n)$. This warp path is $W =$ (1,1), (2,1), (3,1), (4,2), (5,3), (6,4), (7,5), (8,6), (9,7), (9,8), (9,9), (9,10), (10,11), (10,12), (11,13), (12,14), (13,15), (14,15), (15,15), (16,16). If the warp path passes through a cell (i, j) in the matrix, it means that the i'th point in time series X is warped to the j'th point in time series Y.

Since the value at $D(i, j)$ is the minimum warp distance of two time series of lengths i and j, using the dynamic programming approach we can calculate the minimum warp distances starting from $D(1, 1)$ and expanding to all portions of the sequences. Since the warp past must either be incremented by one or stay the same along the i and j axes, the distances of the optimal warp paths one data point smaller than lengths i and j are contained in the matrix at $D(i - 1, j)$, $D(i, j - 1)$, and $D(i - 1, j - 1)$. So the value of a cell in the cost matrix is:

$$D(i, j) = \text{dist}(x_i, y_j) + \min[D(i - 1, j)\,,\, D(i, j - 1)\,,\, D(i - 1, j - 1)] \tag{5.4}$$

After all elements of the matrix are calculated, a warp path must be found from $D(1, 1)$ to $D(m, n)$. The warp path is calculated in reverse order starting at $D(m, n)$ and stopping when $D(1, 1)$ is reached. A greedy search is performed that evaluates cells to the left, down, and diagonally to the bottom-left.

The DTW time and space complexity is $O(mn)$, that is, a quadratic complexity. This complexity is prohibitive for larger sequences, and other approaches can be

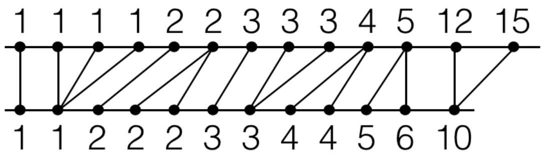


Figure 5.4: A warping between two ordered degree sequences using the function (5.5) to calculate the distances between the degrees. The distance between the two sequences, that is the sum of costs of matched elements, is 0.9.

used to approximate the DTW (with linear time and space complexity) [43].

5.2.2 Using DTW to compare degree sequences

We will use DTW to compare ordered degree sequences. Since elements of sequences A and B are degrees of nodes, we adopt the following distance function:

$$\text{dist}(a,b) = \frac{\max(a,b)}{\min(a,b)} - 1 \tag{5.5}$$

Note that when $a = b$ then $\text{dist}(a,b) = 0$. Thus, two identical ordered degree sequences will have zero distance. Also note that by taking the ratio between the maximum and the minimum, the degrees 1 and 2 are much more different than degrees 101 and 102, a desired property when measuring the distance between node degrees. Figure 5.4 shows DTW applied to two ordered degree sequences.

Last, the function g in equation (5.1) is simply replaced by DTW. Note that k plays a key role in determining the structural distance between two nodes: $f_0(u,v) = 0$ if degrees of u and v are identical, while if $f_{k^*}(u,v) = 0$ then there is strong evidence that there exists an automorphism in G that maps u to v, since the degree sequence of all k-hop rings around u and v perfectly match. Note that if indeed there exists an automorphism in G that maps u to v, then $f_k(u,v) = 0$, for all k. Thus, structural similarity between u and v becomes more rigid as k increases.

5.3 Constructing the multilayer graph

We construct a multilayer weighted graph that encodes structural similarity between nodes. Recall that $G = (V,E)$ denotes the original network (possibly not connected) and k^* its diameter. Let M denote the multilayer graph, with layers going from 0 to k^*, corresponding to neighborhood hierarchy defined above. In particular, layer k will be defined using the k-hop neighborhoods of the nodes in V.

Each layer $k = 0, \ldots, k^*$ is formed by a weighted undirected complete graph with node set V, and thus, $\binom{n}{2}$ edges. The edge weight between two nodes in given layer is given by:

$$w_k(u, v) = e^{-f_k(u,v)}, \quad k = 0, \ldots, k^* \tag{5.6}$$

Note that edges are defined only if $f_k(u, v)$ is defined and that weights are inversely proportional to structural distance, and assume values smaller than or equal to 1, being equal to 1 only if $f_k(u, v) = 0$. Note that nodes that are structurally similar to u will have larger weights across various layers of M.

We connect the layers using directed edges as follows. Each vertex is connected to its corresponding vertex in the layer above and below (layer permitting). Thus, every vertex $u \in V$ in layer k is connected to the corresponding vertex u in layer $k + 1$ and $k - 1$. The edge weight between layers are as follows:

$$\begin{aligned} &w(u_k, u_{k+1}) = \log(\Gamma_k(u) + e), \quad k = 0, \ldots, k^* - 1 \\ &w(u_k, u_{k-1}) = 1, \quad k = 1, \ldots, k^* \end{aligned} \tag{5.7}$$

where $\Gamma_k(u)$ is number of edges incident to u that have weight larger than the average edge weight of the complete graph in layer k. In particular:

$$\Gamma_{k(u)} = \sum_{v \in V} \mathbb{1}(w_{k(u,v)} > \overline{w_k}) \tag{5.8}$$

where $\overline{w_k} = \sum_{(u,v) \in \binom{V}{2}} w_k(u, v) / \binom{n}{2}$. Thus, $\Gamma_k(u)$ measures the similarity of node u to other nodes in layer k. Note that if u has many similar nodes in the current layer, then it should change layers to obtain a more refined context. Note that by moving up one layer the number of similar nodes can only decrease. Last, the log function simply reduces the magnitude of the potentially large number of nodes that are similar to u in a given layer.

Finally, note that M has nk^* vertices and $k^* \binom{n}{2} + 2n(k^* - 1)$ weighted edges. In Section 5.6 we discuss how to reduce the complexity of generating and storing M.

5.4 Generating context for vertices

The multilayer graph M is used to generate structural context for each node $u \in V$. Note that M captures the structure of structural similarities between nodes in G using absolutely no label information. As in previous works, *struct2vec* uses random walks to generate sequence of nodes to determine the context of a given node. In particular, we consider a weighted random walk that moves around M making random choices according to the weights of M. Before each step, the random walk first decides if it will change layers or walk on the current layer. In particular, with probability $q > 0$ the random walk decides to stay in the current layer.

Given that it will stay in the current layer, the probability of stepping from node

u to node v in layer k is given by:

$$p_k(u, v) = \frac{e^{-f_k(u,v)}}{Z_k(u)} \tag{5.9}$$

where $Z_k(u)$ is the normalization factor for vertex u in layer k, simply given by:

$$Z_k(u) = \sum_{\substack{v \in V \\ v \neq u}} e^{-f_k(u,v)} \tag{5.10}$$

Note that the random walk will prefer to step onto vertices that are structurally more similar to the current vertex, avoiding vertices that have very little structural similarity with the current vertex. Thus, the context of a node $u \in V$ is likely to have structurally similar nodes, independent of their labels and position on the original network G.

With probability $1 - q$, the random walk decides to change layers, and moves to corresponding node either in layer $k + 1$ or layer $k - 1$ (layer permitting) with probability proportional to the edge weights. In particular:

$$\begin{aligned} p_k(u_k, u_{k+1}) &= \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})} \\ p_k(u_k, u_{k-1}) &= 1 - p_k(u_k, u_{k+1}) \end{aligned} \tag{5.11}$$

Also important, every time the walker steps within a layer it generates its current position as a vertex of V, independent of the layer. Thus, a vertex u may have a given context in layer k (determined by the structural similarity of this layer), but have a subset of this context at layer $k + 1$, as the structural similarity cannot increase as we move to higher layers. This notion of a hierarchical context across the layers is a fundamental aspect of the proposed methodology.

Finally, for each node $u \in V$, we start a random walk in its corresponding vertex in layer 0. Random walks have a fixed and relatively short length t (number of steps), and the process is repeated a certain number of times (γ), giving rise to multiple independent walks. These node sequences generated by these walks form the context of node u.

5.5 Learning a language model

Skip-Gram [28] has proven to be effective at learning meaningful representations for a variety of data. In order to apply it to networks, it suffices to use artificially generated node sequences instead of word sentences. In our framework, we train the neural network Skip-gram using hierarchical softmax according to optimization problem given by equation (3.22d), using node sequences as training data. These node sequences are generated by biased random walks that have moved around the multilayer graph M. Negative Sampling also can be used as a technique to approximate the softmax function used at the last layer of the Skip-gram.

To train the Skip-gram, it is necessary to define the size of the window (w) to be used as context and the dimension (d) of the node representations. The window is the maximum distance between the current and predicted node within a sequence. Then, given the representation of a node v, we would like to maximize the probability of neighbors of v in the walk (inside w). Note that, because the edge weights, the random walks prefer to step onto vertices with similar structure, so the neighbors of v should be structurally similar to v.

In this phase, after training the Skip-Gram, node latent representations that captures the structural equivalence of nodes will have been generated. Finally, note that while we use Skip-gram to learn node embeddings, virtually any technique to learn representations for text data could be used in its place.

5.6 Computational complexity

In order to construct M, the structural distance between every node pair for every layer must be computed, namely, $f_k(u, v)$ for $u, v \in V$, and $0 \leq k \leq k^*$. However, each value of $f_k(u, v)$ uses the result of the DTW calculation between two degree sequences. While classic implementation of DTW has complexity $O(\ell^2)$, fast techniques have complexity $O(\ell)$, where ℓ is the size of the largest sequence [43]. Let $d_{\max}$ denote the largest degree in the network. Then, the size of the degree sequence $|s(R_k(u))| \leq \min(d_{\max}^k, n)$, for any node u and layer k. Since in each layer there are $\binom{n}{2}$ pairs, the complexity of computing all distances for layer k is $O(n^2 \min(d_{\max}^k, n))$. The final complexity is then $O(k^* n^3)$. In what follows we describe a series of optimizations that will significantly reduce the computation and memory requirements of the framework.

5.6.1 Reducing the length of degree sequences (OPT1)

Although degree sequences at layer k have lengths bounded by $\min(d_{\max}^k, n)$, for some networks this can be quite large even for small k (e.g., for $k = 3$ the sequences are already $O(n)$). Figure 5.5(a) shows the distance distributions of the size of ordered degree sequences generated using the BlogCatalog network [37]. The network has 10,312 nodes, 333,983 edges and the diameter is 5. The degree sequences have many degrees repeated, making their size increase considerably. The layers two and three are the layers having larger sizes of sequences, having sequences with until 9,600

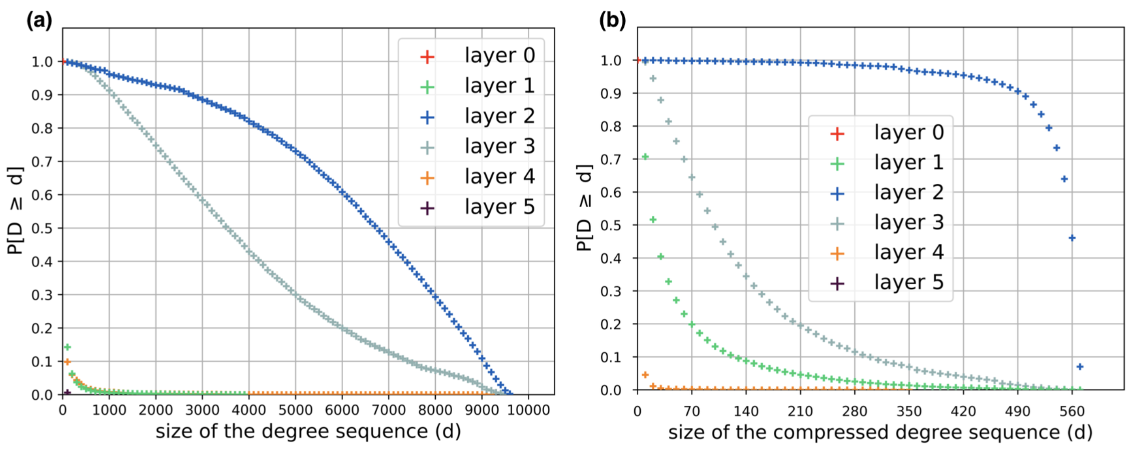


Figure 5.5: Distance distributions of (a) the size of ordered degree sequences and (b) the size of compressed ordered degree sequences, both of the BlogCatalog network [37].

node degrees. The layers one and five have small degree sequences, because, in layer one the sequences have the size of the vertex degree and the layer five is bounded by the diameter of the network, having few degrees.

To reduce the cost of comparing large sequences, we propose compressing the ordered degree sequence as follows. For each degree in the sequence, we count the number of occurrences of that degree. The compressed ordered degree sequence is composed of tuples with the degree and the number of occurrences. Since many nodes in a network tend to have the same degree, in practice the compressed ordered degree sequence can be an order of magnitude smaller than the original. Figure 5.5(b) shows the distance distributions of the size of compressed ordered degree sequences.

Let A' and B' denote the compressed degree sequences of A and B, respectively. Since the elements of A' and B' are tuples, we adapt the DTW pairwise distance function as follows:

$$\mathrm{dist}'(\boldsymbol{a}, \boldsymbol{b}) = \left(\frac{\max(a_0, b_0)}{\min(a_0, b_0)} - 1\right) \max(a_1, b_1) \tag{5.12}$$

where $\boldsymbol{a} = (a_0, a_1)$ and $\boldsymbol{b} = (b_0, b_1)$ are tuples in A' and B', respectively; a_0 and b_0 are the degrees; a_1 and b_1 are the number of occurrences. Note that using the compressed degree sequence leads to comparisons between pieces of the original sequences that have the same degree (as opposed to comparing every degree).

Consider the difference between the DTW distances calculated with original distance function (5.5) and the new distance function (5.12), measured as $|\mathrm{dist}(a, b) - \mathrm{dist}'(\boldsymbol{a}, \boldsymbol{b})|$. We measure the difference distribution between the two distances calculated for all vertex pairs in BlogCatalog network (see Figure 5.6a).

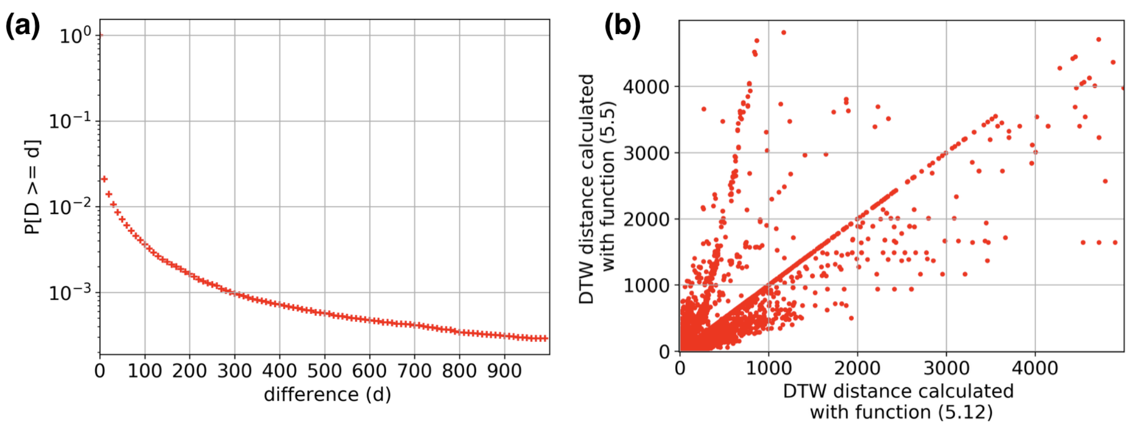


Figure 5.6: (a) Difference distribution between the DTW distances calculated with original distance function (5.5) and the new distance function (5.12). (b) Scatter plot of the DTW distances calculated with the functions (5.5) and (5.12) (each point correspond to a vertex pair of the BlogCatalog network [37]).

Only 2% of new DTW distances have a difference larger than 10. Figure 5.6b shows the correlation between the two DTW distances (calculated with the functions (5.5) and (5.12)) through scatter plot, showing that distances are strongly correlated.

Thus, equation (5.12) leads to an approximation of the DTW on the original degree sequences, as given by equation (5.5). However, DTW now operates on A' and B', which are much shorter than A and B, respectively.

5.6.2 Reducing the number of pairwise similarity calculations (OPT2)

While the original framework assesses the similarity between every node pair at every layer k, clearly this seems unnecessary. Consider two nodes with very different degrees (eg., 2 and 20). Their structural distance even for $k = 0$ will be large, and consequently the edge between them in M will have a very small weight. Thus, when generating context for these nodes, the random walk is unlikely to traverse this edge. Consequently, not having this edge in M will not significantly change the model.

We limit the number of pairwise similarity calculations to $\Theta(\log n)$ per node, for every level k. Let J_u denote the set of nodes that will be neighbors of u in M, which will be the same for every level. J_u should have the nodes most structurally similar to u. In order to determine J_u, we take the nodes that have degrees most similar to u. This can be computed efficiently by performing a binary search on the ordered degree sequence of all nodes in the network (for the degree of node u), and taking $\log n$ consecutive nodes on each direction after the search completes. Thus, computing J_u has complexity $\Theta(\log n)$. Computing J_u for all nodes has complexity

$\Theta(n \log n)$ which is also needed for sorting the degrees of the network. As for memory requirements, each layer of M will now have $\Theta(n \log n)$ edges as opposed to $\Theta(n^2)$.

5.6.3 Reducing the number of layers (OPT3)

The number of layers in M is given by the diameter of the network, k^*. However, for many networks the diameter can be much larger than the average distance. Moreover, the importance of assessing the structural similarity between two nodes diminishes with arbitrarily large values for k. In particular, when k is near k^* the length of the degree sequences of the rings become relatively short, and thus $f_k(u, v)$ is not much different from $f_{k-1}(u, v)$. Therefore, we cap the number the layers in M to a fixed constant $k' < k^*$, capturing the most important layers for assessing structural similarity. This significantly reduces the computational and memory requirements for constructing M.

Although the combination of the above optimizations affects the capacity of the framework in generating good representations for nodes that are structurally similar, we will show that their impact is marginal and sometimes even beneficial. Thus, the benefits in reducing computational and memory requirements of the framework greatly outweighs its drawbacks.

Last, we make *struc2vec* available at: `https://github.com/leoribeiro/struc2vec`

Chapter 6

Experimental Evaluation

In what follows we evaluate *struct2vec* in different scenarios in order to illustrate its potential in capturing the structural identity of nodes, also in light of state-of-the-art techniques for learning node representations.

6.1 Barbell graph

We denote $B(h, k)$ as the (h, k)-barbell graph, which can be obtained by connecting two complete graphs K_1 and K_2 (each having h nodes) through a path graph P of length k. We choose two random nodes $b_1 \in V(K_1)$ and $b_2 \in V(K_2)$ to act as the bridges. Using $\{p_1, \ldots, p_k\}$ to denote $V(P)$, we connect b_1 to p_1 and b_2 to p_k, thus joining the three graphs.

We use this specific network to illustrate how *struct2vec* works, since it has a significant number of nodes with the same structural identity. Let $C_1 = V(K_1) \backslash \{b_1\}$ and $C_2 = V(K_2) \backslash \{b_2\}$. Note that all nodes $v \in \{C_1 \cup C_2\}$ are structurally equivalent, in the strong sense that there exists an automorphism that maps one node to the other. Additionally, we also have that all node pairs $\{p_i, p_{k-i}\}$, for $1 \leq i \leq k-1$, along with the pair $\{b_1, b_2\}$, are structurally equivalent in the same strong sense. Figure 6.1 illustrates a $B(10, 10)$ network, where structurally equivalent nodes have

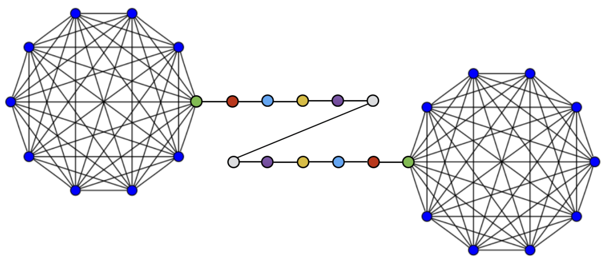


Figure 6.1: Barbell Graph $B(10, 10)$, composed of two complete graphs K with 10 nodes each and a path graph P of length 10.

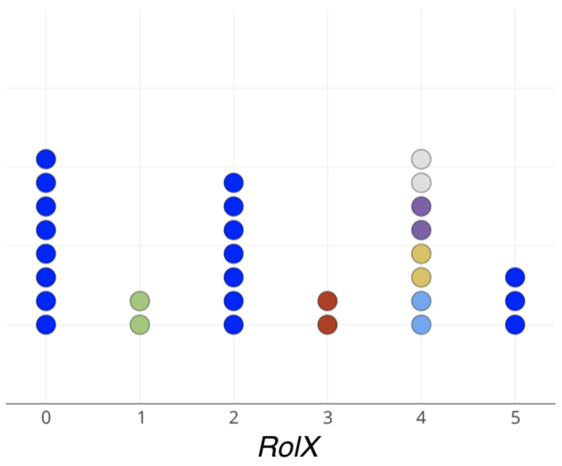


Figure 6.2: Roles identified in Barbell graph $B(10, 10)$ by *RolX*.

the same color.

Thus, we expect *struct2vec* to learn vertex representations that capture the structural equivalence mentioned above. Every node pair that is structurally equivalent should be mapped to points that are close in the latent space. Moreover, the learned representations should also capture structural hierarchies: while the node p_1 is not equivalent to neither nodes p_2 or b_1, we can clearly see that from a structural point of view it is more similar p_2 (it suffices to compare their degrees).

Figure 6.3 shows the latent representations learned by *DeepWalk*, *node2vec* and *struct2vec* (with its optimizations) for the graph $B(10, 10)$. *DeepWalk* fails to capture structural equivalences, which is expected since it was not designed to consider structural identities. As illustrated, *node2vec* does not capture structural identities even with different variations of its parameters p and q. In fact, it learns mostly graph distances, placing closer in the latent space nodes that are closer (in hops) in the graph. Another limitation of *node2vec* is that Skip-gram's window size makes it impossible for nodes from K_1 and K_2 to appear in the same context.

struct2vec, on the other hand, learns representations that properly separate the equivalent classes, and also maps structurally equivalent nodes (in the strong senses) to similar points in the latent space. Note that nodes of the same color are tightly grouped together. Moreover, p_1 and p_{10} are placed close to representations for nodes in K_1 and K_2, as they are the bridges. Finally, note that none of the three optimizations have any significant effect on the quality of the representations. In fact, structurally equivalent nodes are even closer to one another in the latent representations under OPT1.

Last, we apply *RolX* to the barbell graph (results in Figure 6.2). A total of six roles were identified and some roles indeed precisely captured structural equivalence (roles 1 and 3). However, structurally equivalent nodes (in K_1 and K_2) were placed

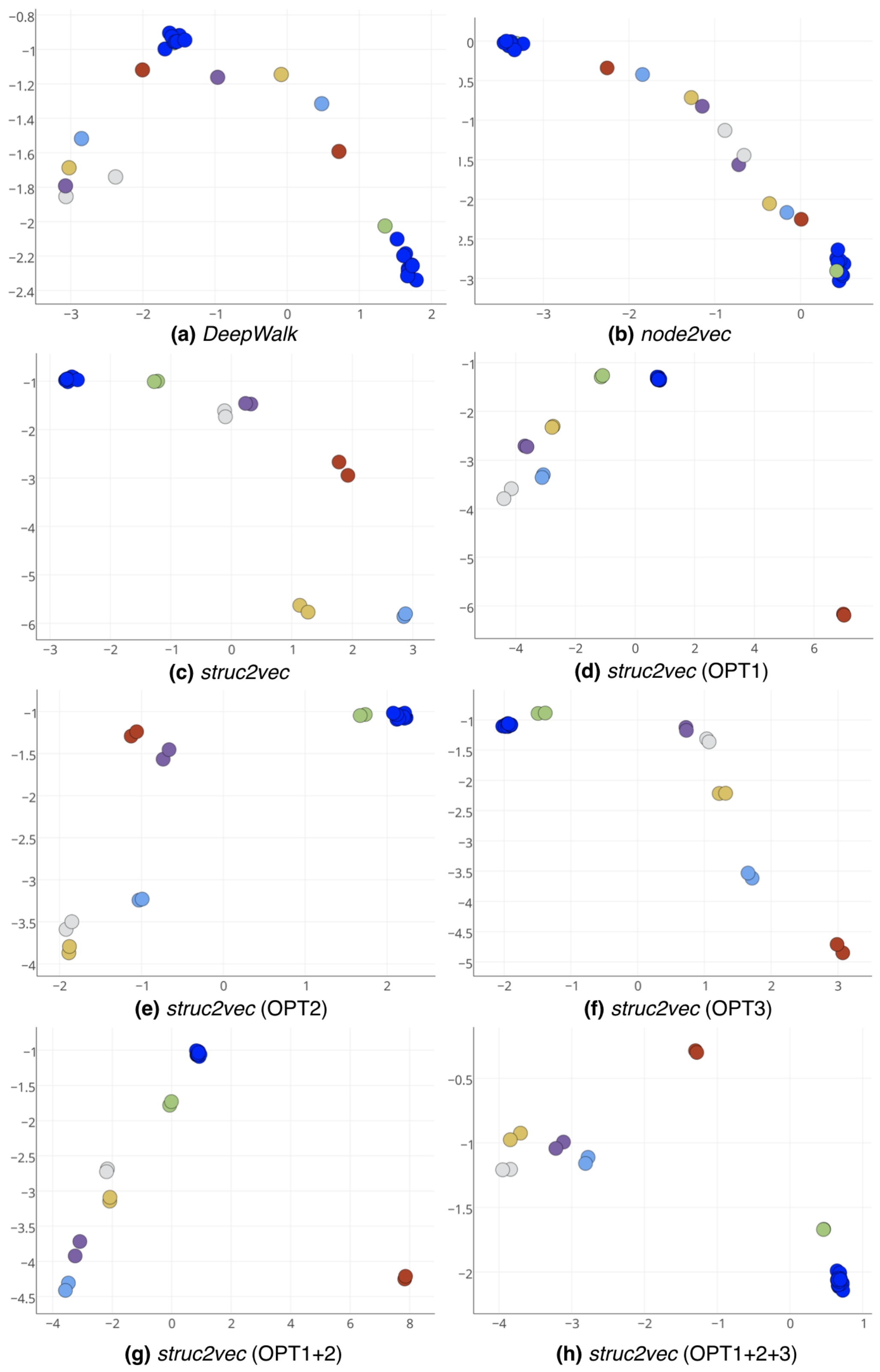


Figure 6.3: Latent representations in $\mathbb{R}^2$ learned by (a) DeepWalk, (b) *node2vec* and (c,d,e,f,g,h) *struc2vec*. Parameters used for all methods: number of walks per node: 20, walk length: 80, skip-gram window size: 5, dimensions: 2. For *node2vec*: $p = 1$ and $q = 2$.

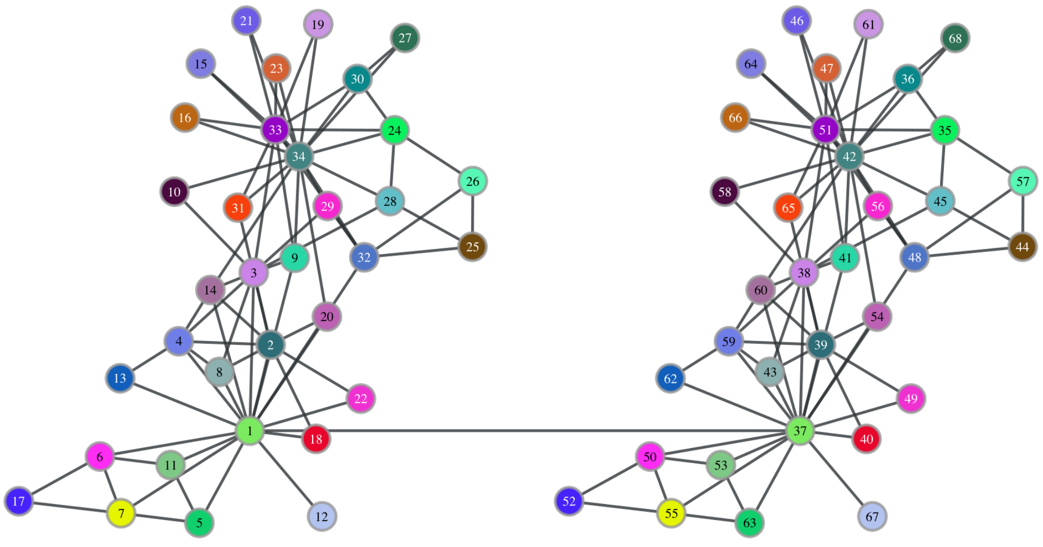


Figure 6.4: Mirrored Karate network. Colors correspond to mirrored nodes.

in three different roles (role 0, 2, and 5) while role 4 contains all remaining nodes in the path. Thus, although *RolX* does capture some notion of structural equivalence when assigning roles to nodes, *struct2vec* better identifies and separates structural equivalence.

6.2 Karate network

The Zachary's Karate Club [35] is an unweighted undirected network composed of 34 nodes and and 78 edges, where each node represents a club member and edges denote if two members have interacted outside the club. In this network, edges are commonly treated as indications of friendship between members.

We construct a graph composed of two copies G_1 and G_2 of the Karate Club network, where each node $v \in V(G_1)$ is a mirrored version of a node $u \in V(G_2)$. We also connect the two networks by adding an edge between mirrored node pairs 1 and 37. Although this is not necessary for our framework, *DeepWalk* and *node2vec* cannot place in the same context nodes in different connected components of the graph. Thus, we add the edge for a more fair comparison with the two baselines. Figure 6.4 shows the generated graph with mirrored node pairs exhibiting the same color.

Figure 6.5 shows the representations learned by *DeepWalk* and *node2vec*, and Figure 6.6a shows the representations learned by *struct2vec*. Clearly, *Deepwalk* and *node2vec* fail to group in the latent space structurally equivalent nodes, as was the case for the Barbell graph, including mirrored nodes.

Once again, *struct2vec* manages to learn features that properly capture the structural identity of nodes. Mirrored pairs – that is, nodes with the same color – stay close together in the latent space, and there is a complex structural hierarchy in the way the representations are grouped together.

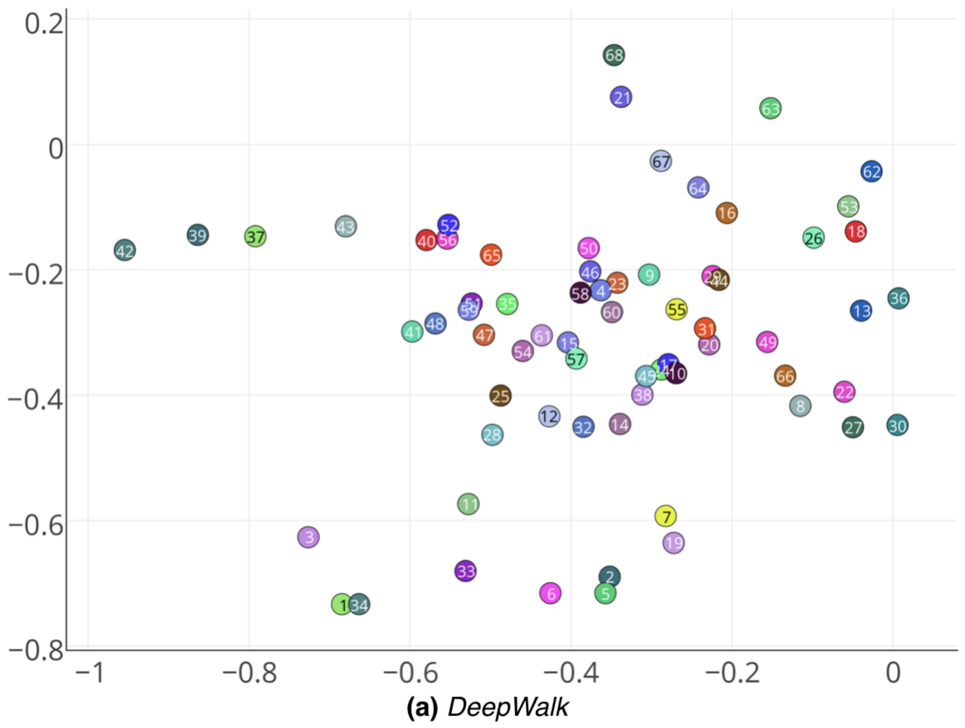
As an example, note that nodes 1, 34 and their correspondent mirrors (37 and 42) are in a separate cluster in the latent space. Interestingly, these are exactly the nodes that represent the club instructor Mr. Hi and his administrator John A. The network was gathered after a conflict between the two that split the members of the club which formed two groups – each centered on either Mr. Hi or John A. Therefore, nodes 1 and 34 have a truly specific – although similar – social role in the original network: they both act as leaders. Note that *struct2vec* captures their function even though there is no edge between them.

Another visible cluster in the latent space is composed of nodes $2, 3, 4$ and 33, also along with their mirrors. These nodes also have a specific structural identity in the network: all of them have high degrees and are also connected to at least one of the leaders. Lastly, nodes 26 and 25 (far right in the latent space) have extremely close representations, which agrees with their structural role: both have low degree and are 2 hops away from leader 34.

struct2vec also captures non-trivial structural equivalences. Note that nodes 7 and 50 (pink and yellow) are mapped to close points in the latent space. Surprisingly, these two nodes are structurally equivalent – there exists an automorphism in the graph that maps one into the other. This can be more easily seen once we note that nodes 6 and 7 are also structurally equivalent, and 50 is the mirrored version of node 6 (therefore also structurally equivalent).

Analyzing how linear transformations in the latent space impact a node's structural identity is fundamental to further understand the learned manifold. Unlike *DeepWalk* and *node2vec*, our technique generates a latent space with a strongly dominant component: clearly, most nodes are spread among a line in the feature space. Note that linearity in this manifold has a direct correspondence to structural properties such as degree. For example, note that $\phi(42) - \phi(3) \approx \phi(3) - \phi(56)$ (where $\phi(i)$ is the latent representation of node i). This suggests that there is a *structural transformation* that maps node 56 to 3, and node 3 to 42. Indeed, it suffices to check each node's degree: $d(42) = 17, d(3) = 10$, $d(56) = 3$, and $d(42) - d(3) = 7 = d(3) - d(56)$. This is a strong indication that the latent space learned by *struc2vec* has fundamental aspects of the structural identity of nodes.

Last, Figure 6.6b shows the roles identified by *RolX* in the mirrored Karate network (28 roles were identified). Note that leaders 1 and 34 were placed in different roles. The mirror for 1 (node 37) was also placed in a different role, while the mirror for 34 (node 42) was placed in the same role as 34. A total of 7 corresponding pairs (out of 34) were placed in the same role. However, some other structural



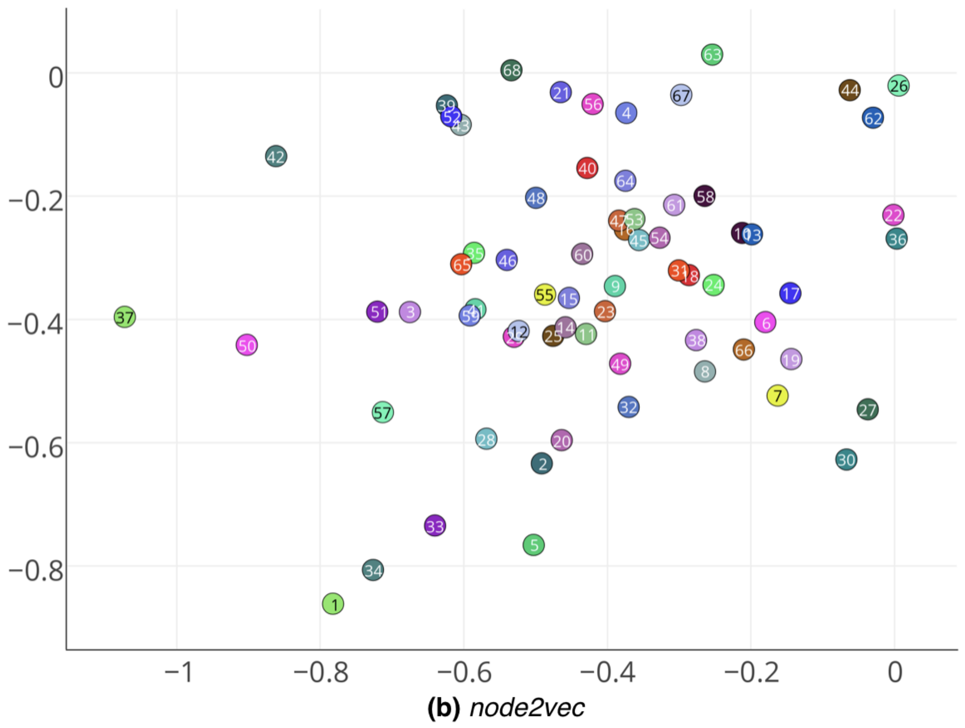
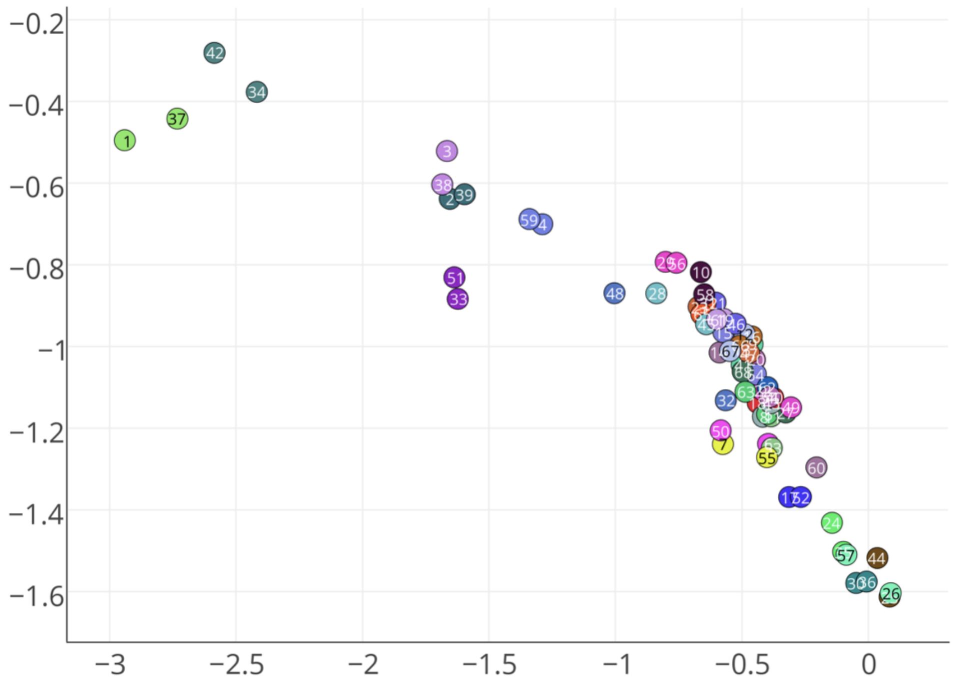
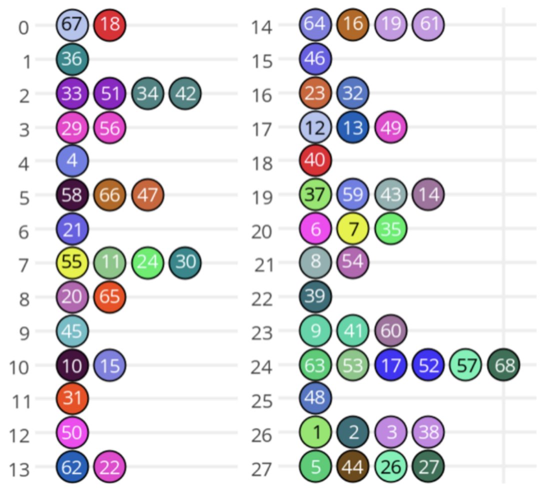


Figure 6.5: Mirrored Karate network representations created by (a) *DeepWalk* and (b) *node2vec*. Parameters used for two methods: number of walks per node: 5, walk length: 15, window size of skip-gram: 3, dimensions: 2. For *node2vec* were used $p = 1$ and $q = 2$.



(a) *struc2vec*



(b) *RolX*

Figure 6.6: (a) Mirrored Karate network representations created by *struc2vec*. Parameters used for the method: number of walks per node: 5, walk length: 15, window size of skip-gram: 3, dimensions: 2. *struc2vec* clearly identifies structurally equivalent nodes (mirrored nodes, with the same color) in the latent space. (b) Roles identified in Mirrored Karate network by *RolX*.

Table 6.1: Average and standard deviation for distances between node pairs in the latent space representation for the mirrored Karate network (see corresponding distributions in Figure 6.7).

Algorithms	Corresponding nodes avg (std)	All nodes avg (std)
DeepWalk	0.377 (0.184)	0.356 (0.195)
node2vec	0.407 (0.199)	0.372 (0.206)
struc2vec	0.129 (0.109)	0.722 (0.694)

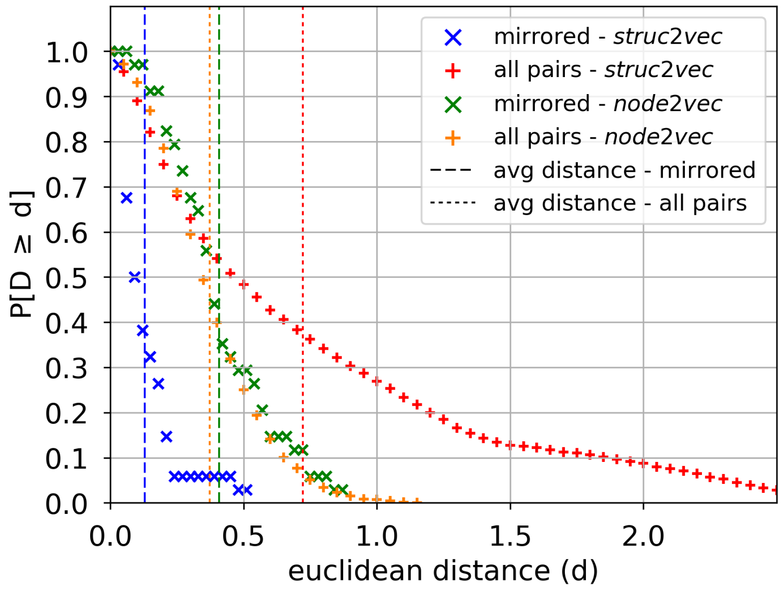


Figure 6.7: Distance distributions between node pairs (mirrored pairs and all pairs) in the latent space, for the mirrored Karate network learned by *node2vec* and *struc2vec* (as shown in Figures 6.6 and 6.5). Curves marked with × correspond to distances between mirrored pairs while + corresponds to all pairs; corresponding averages indicated by vertical lines.

similarities were also identified – e.g., nodes 6 and 7 are structurally equivalent and were assigned the same role. Again, *RolX* seems to capture some notion of structural similarities among network nodes but *struct2vec* can better identify and separate structural equivalences using latent representations.

Consider the distance between pairs of vertices in the latent representation. We measure the distance distribution between pairs corresponding to mirrored nodes and the distance distribution among all node pairs (using the representation shown in Figures 6.6 and 6.5). Figure 6.7 shows the two distance distributions for the representations learned by *node2vec* and *struc2vec*, with corresponding averages shown in Table 6.1. For *node2vec* the two distributions are practically identical, indicating no difference between distances among mirrored pairs and distances among all pairs.

DeepWalk shows similar behavior (curves omitted for clarity) with averages shown in Table 6.1. In contrast, *struc2vec* exhibits two very different distributions: 94% of mirrored node pairs have distance smaller than 0.25 while 68% of all node pairs have distance larger than 0.25. Moreover, the average distance between all node pairs is 5.6 times larger than that of mirrored pairs, while this ratio is about slightly *smaller* than 1 for *DeepWalk* and *node2vec* (see Table 6.1).

To better characterize the relationship between structural distance and distances in the latent representation learned by *struc2vec*, we compute the correlation between the two distances for all node pairs. In particular, for each layer k we compute the Spearman and Pearson correlation coefficients between $f_k(u, v)$, as given by equation (5.1), and the euclidean distance between u and v in the latent representation. Results shown in Table 6.2 for the mirrored Karate network indeed corroborate that there is a very strong correlation between the two distances, for every layer, and captured by both coefficients. This suggests that *struc2vec* indeed captures in the latent space the measure for structural similarity adopted by the methodology.

Table 6.2: Pearson and Spearman correlation coefficients between structural distance and euclidean distance in latent space for all node pairs in the mirrored Karate network

Layer	Pearson correlation (p-value)	Spearman correlation (p-value)
0	0.83 (0.0)	0.74 (0.0)
1	0.72 (0.0)	0.66 (0.0)
2	0.71 (0.0)	0.65 (0.0)
3	0.70 (0.0)	0.59 (0.0)
4	0.70 (0.0)	0.57 (0.0)
5	0.62 (0.0)	0.47 (2.40)
6	0.74 (0.0)	0.57 (2.37)
7	0.91 (0.0)	0.89 (2.45)

6.3 Robustness to edge removal

We consider another scenario to illustrate the potential of the framework in effectively representing structural identity, even in the presence of noise. In particular, we randomly remove edges from the network, directly changing its structure. We adopt the parsimonious *edge sampling model* to instantiate two structurally correlated networks that were subjected to random edge removal [46].

The model works as follows. Starting from a fixed graph $G = (V, E)$, we generate a graph G_1 by sampling each edge $e \in E$ with probability s, independently. Thus, each edge of G is present in G_1 with probability s. Repeat the process again using G to generate another graph G_2. Thus, G_1 and G_2 are structurally correlated through G, and s controls the amount of structural correlation. Note that when $s = 1$, G_1 and G_2 are isomorphic, while when $s = 0$ all structural identity is lost.

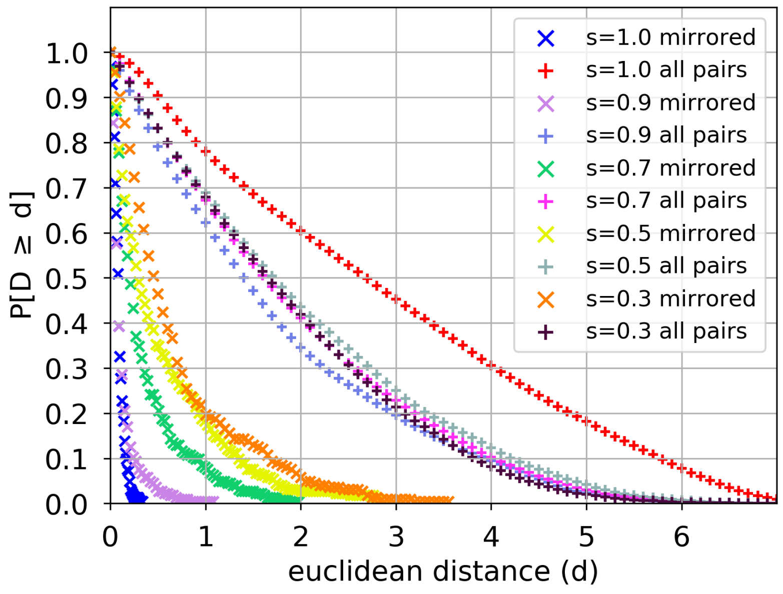


Figure 6.8: Distribution for distances between node pairs in latent space representation, under the edge sampling model (different values for s). Bottom curves (marked with $\times$) are distances between corresponding node pairs; top curves (marked with $+$) are distances between all node pairs.

We apply the edge sampling model to an *egonet* extracted from Facebook (224 nodes, 3192 edges, max degree 99, min degree 1) [47] to generate G_1 and G_2 with different values for s. We relabel the nodes in G_2 (as with the previous example), and consider the union of the two graphs as the input network to our framework. Note that this graph has at least two connected components (corresponding to G_1 and G_2) and every node in G_1 has a corresponding node in G_2 (and vice-versa).

Figure 6.8 shows the distance distribution between node pairs in the latent space under various values for s (corresponding averages are shown in Table 6.3). In order to evaluate how well *struct2vec* captures structural identities in this setting, we compare the distance distributions between all node pairs and between only correspondent pairs.

For $s = 1$ (thus, G_1 is isomorphic to G_2), the two distance distributions are strikingly different, with the average distance for all pairs being 21 times larger than that for corresponding pairs (see Table 6.3). More interestingly, when $s = 0.9$ the two distributions are still very different. Note that while further decreasing s does not significantly affect the distance distribution of all pairs, it slowly increases

Table 6.3: Average and standard deviation for distances between node pairs in the latent space representation (see corresponding distributions in Figure 6.8)

	Corresponding nodes	All nodes
s	avg (std)	avg (std)
1.0	0.083 (0.05)	1.780 (1.354)
0.9	0.117 (0.142)	1.769 (1.395)
0.7	0.338 (0.374)	1.975 (1.438)
0.5	0.528 (0.588)	1.994 (1.480)
0.3	0.674 (0.662)	1.962 (1.445)

the distribution of corresponding pairs. However, even when $s = 0.3$ (which means that the probability that an original edge appears both in G_1 and G_2 is 0.09, s^2), the framework still places together corresponding nodes in the latent space.

This experiment indicates the robustness of the framework in uncovering the structural identity of nodes even in the presence of structural noise, modeled here through edge removals.

6.4 Classification

A common application of latent representations for network nodes is classification. *struc2vec* can be leveraged for this task when labels for nodes are more related to their structural identity than to the labels of their neighbors. To illustrate this potential, we consider air-traffic networks: unweighted, undirected networks where nodes correspond to airports and edges indicate the existence of commercial flights.

We consider the following datasets (collected for this study):

- *Brazilian Air-traffic network:* Data collected from the National Civil Aviation Agency[1], the Brazilian civil aviation authority, and were collected from January to December 2016. The network has 131 vertices, 1038 edges and the diameter is 5. We label the airports taking into account their movements in Brazil, during the year 2016. A movement is a landing or takeoff of an aircraft. Airport activity is measured by the total number of movements in the corresponding year.

- *American air-traffic network:* Data collected from the Bureau of Transportation Statistics[2] from January to October, 2016. The network has 1,190 nodes, 13,599 edges (diameter is 8). Airport activity is measured by the total number

[1]http://www.anac.gov.br/
[2]https://transtats.bts.gov/

Table 6.4: Parameter values used in grid search. Parameters for *node2vec* and *struc2vec*: number of walks per node (γ), walk length (t), context window-size (w) and dimensions (d). Parameters p and q used only by *node2vec*.

network of flights	γ	t	w	d	p , q
Brazil	5, 10, 15, 20, 30	5, 10, 15, 20, 30, 40, 60, 70, 80	1, 2, 3, 5, 10, 20	8, 16, 32, 64	0.25, 0.50, 1, 2, 4
EUA	5, 10, 15, 20, 30, 50	10, 15, 20, 30, 40, 60, 70, 80, 90, 120	2, 3, 5, 10, 20	64, 128, 256	0.25, 0.50, 1, 2, 4
Europe	5, 10, 15, 20, 30	5, 10, 15, 20, 30, 40, 60, 70, 80	1, 2, 3, 5, 10, 20	8, 16, 32, 64, 128	0.25, 0.50, 1, 2, 4

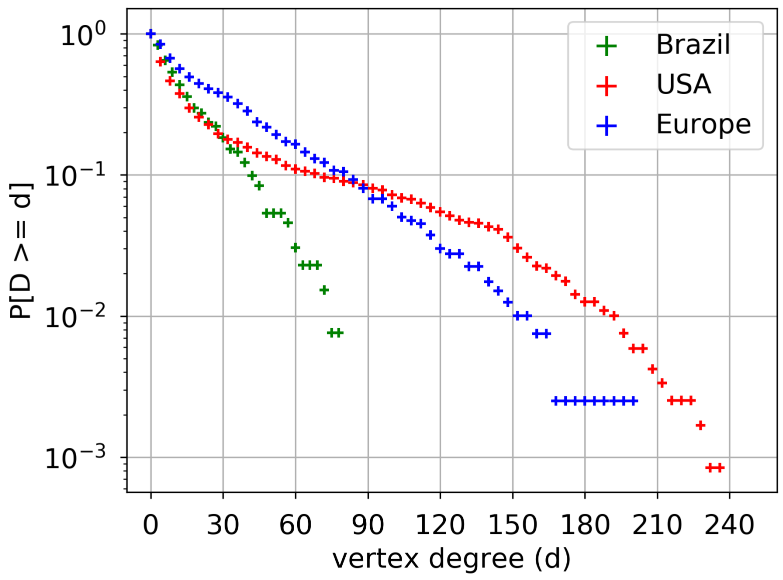


Figure 6.9: Distribution of vertex degrees of air-traffic networks of Brazil, USA and Europe.

of people that passed (arrived plus departed) the airport in the corresponding period.

- *European air-traffic network:* Data collected from the Statistical Office of the European Union (Eurostat)[3] from January to November 2016. The network has 399 nodes, 5,995 edges (diameter is 5). Airport activity is measured by the total number of movements in the corresponding period.

An air-traffic network usually has a heavy-tail degree distribution, with few airports playing the role of major hubs [19]. In a network with a heavy-tail degree distribution most nodes have only a few links. These various small nodes are held

[3]http://ec.europa.eu/

together by a few highly connected hubs. Then, we believe that airports with similar size are probably to have a similar structure in the network. For example, very large airports, like Garulhos (GRU) or Galeão (GIG) in Brazil or Atlanta Airport (ATL) and Los Angeles Airport (LAX) in EUA, will have many flights arriving and departing (movements) and they will act like hubs in the network, performing the role of connecting several smaller airports. At the same time, many airports are tiny and have only a few connections (e.g. degree 1 or 2). They will be on the periphery of the network, being final destinations for people going to small towns or being private airports. Figure 6.9 shows the degree distribution of the networks.

Airports will be assigned a label corresponding to their level of activity, measured in flights or people. For each airport, we assign one of four possible labels corresponding to their activity. In particular, for each dataset, we use the quartiles obtained from the empirical activity distribution to split the dataset in four groups, assigning a different label for each group. Thus, label 1 is given to the 25% less active airports, and so on. Note that all classes (labels) have the same size (number of airports). Moreover, classes are related more to the role played by the airport.

We learn latent representations for nodes of each air-traffic network using *struc2vec* and *node2vec* using a grid search to select the best hyperparameters for each case (see Table 6.4). Note that this step does not use any node label information. The latent representation for each node becomes the feature that is then used to train a supervised classifier (one-vs-rest logistic regression with L2 regularization). We also consider just the node degree as a feature since it captures a very basic notion of structural identity. Last, since classes have identical sizes, we use just the accuracy to assess performance. Experiments are repeated 10 times using random samples to train the classifier (80% of the nodes used for training) and we report on the average performance.

6.4.1 Results

Figure 6.10 shows the classification performance of the different features for all air-traffic networks. Clearly, *struc2vec* outperforms the other approaches, and its optimizations have little influence. As the networks have labels related with the structural equivalence of nodes, *struc2vec* can help to predict airports with structural equivalences.

For the Brazilian network, *struc2vec* improves classification accuracy by 50% in comparison to *node2vec*. Interestingly, for this network *node2vec* has average performance (slightly) inferior to node degree, indicating the importance played by the structural identity of the nodes in classification. Surprisingly, versions of *struc2vec* using optimization 2, have superior performance. We believe that it occurs

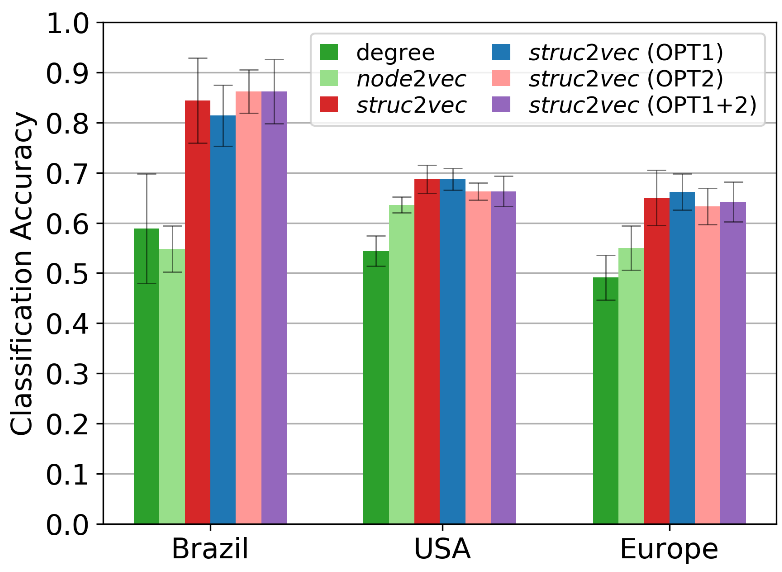


Figure 6.10: Average accuracy for multi-class classification in air-traffic networks of Brazil, USA and Europe for different node features used in supervised learning, with 80% of the nodes labeled for training. The values are the average of 10 executions with random sample initializations.

because the network has many vertex with similar degrees (see Figure 6.9), and forcing compare each network node with only $\Theta(\log n)$ more similar vertices (rather than compare with all other vertices), makes the random walks step between more structurally similar vertices, consequently, refining the context better.

For the American and European air-traffic network, we achieve an improvement over *node2vec* of 8% and 20% in accuracy, respectively. In some cases, using the optimizations proposed in section 5.6 causes the score values to drop slightly. This is expected because the framework will have less information to generate the network used for the generation of vertex contexts.

In American and European air-traffic networks, *node2vec* has a relatively good performance. We believe that this occurred because *node2vec* can generate representations mixing homophily and structural equivalence. As similar airports usually have routes between them, homophily is also present in these networks. For example, large airports, such as ATL, have routes to other major airports, such as LAX, making labels also related to homophily.

Lastly, the labels were generated in a arbitrary way, so airports similar, from the point of view of the number of people who passed through it or of its movements, can be in different class if they are near to the boundary that was used to separate the class. This may have affected the performance of *struc2vec*.

6.5 Scalability

In order to illustrate its scalability, we apply *struc2vec* with the first two optimizations to instances of the Erdös-Rényi random graph model [48]. The following values were used for the parameters: dimensions (d): 128, number of walks per node (γ): 10, walk length (t): 80, Skip-Gram window-size (w): 10.

We compute the average execution time for 10 independent runs on graphs with sizes from 100 to 1,000,000 nodes and average degree of 10. In order to speed up training the language model, we use Skip-Gram with Negative Sampling [29].

To perform the scalability experiments, a computer with the following configurations was used: Processor: Intel(R) Xeon(R) CPU E5-2630, 2.60GHz (24 cores); Memory: 62GB; Operating System: Ubuntu 12.04.5 LTS.

Figure 6.11 shows the execution time, in log-log scale, indicating that *struc2vec* scales super-linearly but closer to linear than to $n^{1.5}$ (dashed lines). The sampling time comprises of calculations of distances between the vertex pairs, the creation of the multi-layer graph M and simulation of random walks. The training time comprises of learning of representations using Skip-Gram.

Thus, despite its unfavorable worst case time and space complexity, in practice *struc2vec* can be applied to very large networks.

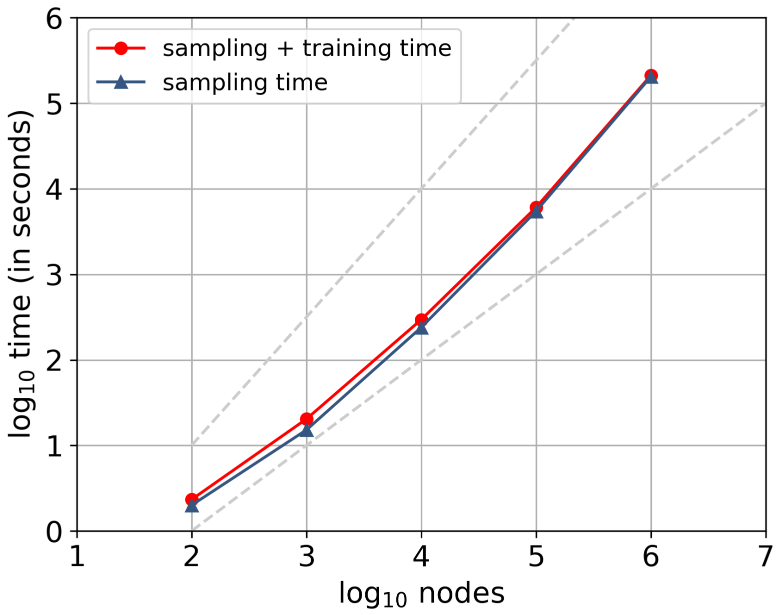


Figure 6.11: Average execution time of *struc2vec* on Erdös-Rényi graphs with average degree of 10. Training time refers to the additional time required by Skip-Gram.

Chapter 7

Conclusion and future work

7.1 Work considerations

Structural identity is a concept of symmetry in networks in which nodes are identified using just the network structure, along with their relationship to other vertices. The concept is strongly related to the notion of function, in which nodes tend to play particular roles in the network. Thus, identifying nodes with similar identity has long been investigated, in social sciences and hard sciences.

In this dissertation, we studied representation learning in networks to capture the structural identity of nodes. Learning representations is a important task in machine learning. Recent works have focused on using neural networks models to build general word representations [29] [27]. A neural language model is a language model based on neural networks, exploiting their ability to learn distributed representations. Recently, *DeepWalk* and *node2vec* brought the representation learning for the context of networks using neural language models. These works propose strategies of learning latent representations for nodes: while *DeepWalk* creates latent representations capturing the homophily present in networks, *node2vec* learns representations pursuing to capture a mixing of homophily and structural equivalence. Even though the experiments suggest that *node2vec* captures structural equivalence, structurally equivalent vertices will never share the same context if their distance (hop count) is larger than the Skip-gram window, being possible for structurally similar vertices to be far in the latent space.

We proposed *struc2vec*, a flexible framework to learn representations that captures the structural identity of nodes in a network. *struc2vec* assesses the structural similarity of node pairs without leveraging node or edge attributes, including node labels. It also uses a hierarchy to measure structural similarity at different scales, using ordered node degree sequence within the k-hop neighborhood node pairs. These structural distances are used to construct a multilayer weighted graph that encodes structural similarities among all nodes in the network. Biased random walks on this multilayer graph are used to generate the structural context for every node.

As we have shown, some techniques fail to learn representations that can effectively capture structural identity, while *struc2vec* overcomes their limitations and excels in this task, in comparison. We applied our framework in different scenarios, demonstrating its effective capacity to capture the structural equivalence of vertices. Specifically, we conducted an experiment with an *egonet* extracted from Facebook, showing the robustness of the framework in uncovering the structural identity of nodes even in the presence of structural noise, modeled through edge removals. Lastly, we executed a classification experiment to find similar airports, based on the network structure. In contrast, *struc2vec* was not designed to capture node *homophily*, a common property in networks that can be leveraged for solving many (supervised) classification task.

7.2 Limitations and future work

There is room for further development on a few directions, and we present some of them worth pursuing:

- *struc2vec* only address the structural identity of nodes. Can structural identity and homophily of nodes be adequately captured by a latent representations? On the one hand, structural identity is a concept independent of network position, while on the other hand, homophily is a concept tied to network proximity. Reconciling these two fundamental aspects of network nodes is an open and active research question.

- To create a ordered sequence degree for each vertex a BFS is required, that is a costly procedure, having complexity $O(|V| + |E|)$ per vertex. Besides that, we have adopted only the vertex degree. Using more node features than just degree (combining them, for example), as closeness or clustering coefficient, could improve the latent representations.

- Using another approach to generate sequences of nodes in a way different from random walks. A possible attempt would be defining a distance function that returns in a stochastic way, sequences of nodes that are structurally similar, or that has a certain property to be captured by the representations in the latent space.

- *struc2vec* was only tested in unweighted undirected networks. A generalization to operate with directed networks or/and weighted networks will increase the range of applications that can benefit from the method.

Bibliography

[1] LORRAIN, F., WHITE, H. C. "Structural equivalence of individuals in social networks", *The Journal of mathematical sociology*, v. 1, n. 1, pp. 49–80, 1971.

[2] SAILER, L. D. "Structural equivalence: Meaning and definition, computation and application", *Social Networks*, v. 1, n. 1, pp. 73–90, 1978.

[3] SINGH, R., XU, J., BERGER, B. "Global alignment of multiple protein interaction networks with application to functional orthology detection", *Proceedings of the National Academy of Sciences*, v. 105, n. 35, pp. 12763–12768, 2008.

[4] ATIAS, N., SHARAN, R. "Comparative analysis of protein networks: hard problems, practical solutions", *Communications of the ACM*, v. 55, n. 5, pp. 88–97, 2012.

[5] PIZARRO, N. "Structural Identity and Equivalence of Individuals in Social Networks Beyond Duality", *International Sociology*, v. 22, n. 6, pp. 767–792, 2007.

[6] KLEINBERG, J. M. "Authoritative sources in a hyperlinked environment", *Journal of the ACM (JACM)*, v. 46, n. 5, pp. 604–632, 1999.

[7] LEICHT, E. A., HOLME, P., NEWMAN, M. E. "Vertex similarity in networks", *Physical Review E*, v. 73, n. 2, pp. 026120, 2006.

[8] FOUSS, F., PIROTTE, A., RENDERS, J.-M., et al. "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation", *IEEE Transactions on knowledge and data engineering*, v. 19, n. 3, 2007.

[9] BLONDEL, V. D., GAJARDO, A., HEYMANS, M., et al. "A measure of similarity between graph vertices: Applications to synonym extraction and web searching", *SIAM review*, v. 46, n. 4, pp. 647–666, 2004.

[10] ZAGER, L. A., VERGHESE, G. C. "Graph similarity scoring and matching", *Applied mathematics letters*, v. 21, n. 1, pp. 86–94, 2008.

[11] HENDERSON, K., GALLAGHER, B., ELIASSI-RAD, T., et al. "Rolx: structural role extraction & mining in large graphs". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1231–1239. ACM, 2012.

[12] GROVER, A., LESKOVEC, J. "node2vec: Scalable Feature Learning for Networks". In: *ACM SIGKDD*, 2016.

[13] NARAYANAN, A., CHANDRAMOHAN, M., CHEN, L., et al. "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs". In: *International Workshop on Mining and Learning with Graphs*, 2016.

[14] PEROZZI, B., AL-RFOU, R., SKIENA, S. "DeepWalk: Online Learning of Social Representations". In: *ACM SIGKDD*, 2014. ISBN: 978-1-4503-2956-9.

[15] TANG, J., QU, M., WANG, M., et al. "LINE: Large-scale Information Network Embedding". In: *WWW*, 2015.

[16] FORTUNATO, S. "Community detection in graphs", *Physics reports*, v. 486, n. 3, pp. 75–174, 2010.

[17] R. RIBEIRO, L. F., P. SAVERESE, P. H., R. FIGUEIREDO, D. "struc2vec: Learning Node Representations from Structural Identity". In: *ACM SIGKDD*, 2017.

[18] NEWMAN, M. "Networks: an introduction. 2010", *United Slates: Oxford University Press Inc., New York*, pp. 1–2.

[19] BARABÁSI, A.-L. "Network science book", *Boston, MA: Center for Complex Network, Northeastern University*, 2014.

[20] DE LAS RIVAS, J., FONTANILLO, C. "Protein–protein interactions essentials: key concepts to building and analyzing interactome networks", *PLoS Comput Biol*, v. 6, n. 6, pp. e1000807, 2010.

[21] WINSHIP, C., MANDEL, M. "Roles and positions: A critique and extension of the blockmodeling approach", *Sociological methodology*, v. 14, pp. 314–344, 1983.

[22] WEST, D. B., OTHERS. *Introduction to graph theory*, v. 2. Prentice hall Upper Saddle River, 2001.

[23] MATHON, R. "A note on the graph isomorphism counting problem", *Information Processing Letters*, v. 8, n. 3, pp. 131–136, 1979.

[24] SIMÕES, J. E. *Two Problems On The Structure-Identity Relationship On Networks*. Tese de Doutorado, Universidade Federal do Rio de Janeiro, 2016.

[25] RONG, X. "word2vec parameter learning explained", *arXiv preprint arXiv:1411.2738*, 2014.

[26] BENGIO, Y., COURVILLE, A., VINCENT, P. "Representation learning: A review and new perspectives", *IEEE transactions on pattern analysis and machine intelligence*, v. 35, n. 8, pp. 1798–1828, 2013.

[27] BENGIO, Y., DUCHARME, R., VINCENT, P., et al. "A neural probabilistic language model", *Journal of machine learning research*, v. 3, n. Feb, pp. 1137–1155, 2003.

[28] MIKOLOV, T., CHEN, K., CORRADO, G., et al. "Efficient Estimation of Word Representations in Vector Space". In: *ICLR Workshop*, 2013.

[29] MIKOLOV, T., SUTSKEVER, I., CHEN, K., et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*, pp. 3111–3119, 2013.

[30] COLLINS, M. "Language Modeling - Course notes for NLP, Columbia University", 2013.

[31] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J. *Learning internal representations by error propagation*. Relatório técnico, DTIC Document, 1985.

[32] MORIN, F., BENGIO, Y. "Hierarchical Probabilistic Neural Network Language Model". In: Cowell, R. G., Ghahramani, Z. (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pp. 246–252, 2005.

[33] GUTMANN, M. U., HYVÄRINEN, A. "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics", *Journal of Machine Learning Research*, v. 13, n. Feb, pp. 307–361, 2012.

[34] SHERVASHIDZE, N., SCHWEITZER, P., VAN LEEUWEN, E. J., et al. "Weisfeiler-Lehman Graph Kernels", *J. Mach. Learn. Res.*, v. 12, pp. 2539–2561, nov. 2011. ISSN: 1532-4435.

[35] ZACHARY, W. W. "An information flow model for conflict and fission in small groups", *Journal of anthropological research*, v. 33, n. 4, pp. 452–473, 1977.

[36] KNUTH, D. E. *The Stanford GraphBase: a platform for combinatorial computing*, v. 37. Addison-Wesley Reading, 1993.

[37] ZAFARANI, R., LIU, H. "Social Computing Data Repository at ASU [http://socialcomputing. asu. edu]. Tempe, AZ: Arizona State University, School of Computing", *Informatics and Decision Systems Engineering*, 2009.

[38] BREITKREUTZ, B.-J., STARK, C., REGULY, T., et al. "The BioGRID interaction database: 2008 update", *Nucleic acids research*, v. 36, n. suppl 1, pp. D637–D640, 2008.

[39] MAHONEY, M. "Large text compression benchmark", *URL: http://www. mattmahoney. net/text/text. html*, 2011.

[40] TANG, L., LIU, H. "Leveraging social media networks for classification", *Data Mining and Knowledge Discovery*, v. 23, n. 3, pp. 447–478, 2011.

[41] TANG, J., QU, M., WANG, M., et al. "Line: Large-scale information network embedding". In: *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. ACM, 2015.

[42] RAKTHANMANON, T., CAMPANA, B., MUEEN, A., et al. "Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping", *ACM Transactions on Knowledge Discovery from Data (TKDD)*, v. 7, n. 3, 2013.

[43] SALVADOR, S., CHAN, P. "FastDTW: Toward accurate dynamic time warping in linear time and space". In: *Workshop on Mining Temporal and Sequential Data, ACM SIGKDD*, 2004.

[44] MÜLLER, M. "Dynamic time warping", *Information retrieval for music and motion*, pp. 69–84, 2007.

[45] SENIN, P. "Dynamic time warping algorithm review", *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, v. 855, pp. 1–23, 2008.

[46] PEDARSANI, P., GROSSGLAUSER, M. "On the privacy of anonymized networks". In: *ACM SIGKDD*, 2011.

[47] LESKOVEC, J., MCAULEY, J. J. "Learning to discover social circles in ego networks". In: *Advances in neural information processing systems*, pp. 539–547, 2012.

[48] ERDÖS, P., RÉNYI, A. "On random graphs, I", *Publicationes Mathematicae (Debrecen)*, v. 6, pp. 290–297, 1959.