UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Network and Cascade Representation Learning

Algorithms based on Information Diffusion Events

Zekarias Tilahun Kefato

Advisor

Prof. Alberto Montresor

Università degli Studi di Trento

April 18, 2019

Dedicated to
my father Tilahun Kefato
and
my mother Elisabeth Ergeno

Abstract

Network representation learning (NRL) and cascade representation learning (CRL) are fundamental backbones of different kinds of network analysis problems. They are usually carried out in settings where the structure of the network under consideration is known. Motivated by real-world problems, this study presents several algorithms for scenarios where the network structure is partially or completely unknown.

The objective of network representation learning is to identify a mapping function that projects sparse and high-dimensional network graphs into a dense latent representation, which preserves the original information about nodes and their neighborhoods. The notion of neighborhood, however, becomes illusive when the network structure is partially or completely hidden.

Inspired by previous results, in our thesis work we have developed novel algorithms that are resilient to such lack of knowledge. These results establish a correlation between the properties of the network and different kind of node activities performed over it, information which is generally more available and can be easily observed. In particular, we focus on diffusion events – also called cascades – such as shares, retweets and hashtags.

In the first of our contributions, we have developed a novel NRL algorithm called Mineral*, a simple technique that combines the observed cascades with the partially accessible network structure by sampling artificial cascades. Node representation is then learned from the observed and sampled cascades by using the* SkipGram *model that is widely used for word rep-*

resentation learning in natural language documents.

In our second contribution, called NetTensor*, we assume that the network structure is completely hidden and we propose novel techniques that are capable to estimate both the hidden neighborhood (proximity) and the similarity of nodes. Such estimated values are then used to learn a unified embedding of nodes using a scalable truncated singular value decomposition and deep autoencoders.*

In addition to the NRL algorithms, we have also proposed a novel CRL algorithm called cas2vec *for virality (popularity) prediction. Again, we pursue a network-agnostic approach following the above assumption that the network structure is completely unknown. Unlike prior studies that rely on manual feature extraction,* cas2vec *automatically learns cascade representations based on convolutional neural networks, that are effective in predicting virality of cascades.*

We have carried out extensive experiments using several real-world datasets for all of our methods and compared them against strong baselines from the state-of-the-art, achieving significantly better results than many of them.

Keywords

[Network representation learning, information diffusion, deep learning, matrix factorization, social network analysis, cascade prediction]

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graphs are widely used to model sets of entities that interact over a given medium, such as users in social networks, blogs over the Internet, proteins in biological networks, locations in road networks, and so on. Efficiently representing the entities (nodes) and their interactions (edges) is a critical step towards performing meaningful analysis over such complex networks.

Graphs are usually represented "as is", using basic data structures such as adjacency or incidence matrices. For example, an adjacency matrix $\mathbf{M} \in [0,1]^{n \times n}$, where n is the number of nodes, is such that $\mathbf{M}[i,j]$ captures the existence (1) or the absence (0) of an edge between node i and j. Despite its simplicity, performing some kinds of tasks (e.g., link prediction or node classification) based on such complete representation usually leads to poor performance; the resulting computation can be very expensive depending on n, commonly known as the curse-of-dimensionality.

For this reason, alternative techniques have been designed to encode graphs in more compact and efficient ways. Consider an encoding oracle $O : [0,1]^{n \times n} \rightarrow \mathbb{R}^{n \times d}$, which transforms a complete representation $\mathbf{M}$ into an encoded representation $\mathbf{Z} = O(\mathbf{M})$, where $d \ll n$.

$\mathbf{Z}$ is a compact and dense representation obtained by applying the oracle O such that the "most important" properties of $\mathbf{M}$ are preserved in $\mathbf{Z}$. The

problem of *network representation learning* (NRL) or graph embedding amounts to identifying the oracle O.

In the last few decades, several strategies have been proposed towards this goal. The classical approaches are based on explicit matrix factorization, such as *latent semantic indexing* (LSI) [14], *non-negative matrix factorization* (NMF) [60, 49], or *singular value decomposition* (SVD).

More recent approaches utilize artificial neural networks and can be subdivided in two main groups. On one hand, some studies seek to preserve the *local* and/or *global* properties of nodes [64, 70, 28, 77, 19, 32, 1, 59, 45, 29, 44]. Potential properties of interest are first-order, second-order, and in general higher-order proximities of nodes, or cohesive structures such as community subgraphs, etc. These approaches are usually suitable to preserve similarities (homophily) between nodes.

On the other hand, some complementary studies have endeavored to preserve the *role* of nodes, such as being hubs, bridges or peripheral nodes [32, 19, 1, 45, 30].

In addition to the aforementioned works, recent studies have been proposed to take advantage of additional aspects of graphs, as well. In the early stage of the neural network representation learning (NNRL), several papers have exploited the network structure only [64, 28, 70, 77]. Follow-up studies have proposed to incorporate node attributes and have empirically proven that such approach leads to a better quality representation [85, 34, 55, 66].

Furthermore, alternative studies [9, 38, 39, 40] have also been proposed to exploit explicit interaction between nodes of a network. In particular, these studies have focused on interactions that led to information diffusion events. The main advantage of these studies is that their methods can be used for predicting the future state of the diffusion events themselves (cascade prediction).

The focus of this thesis is on the use of information diffusion events in order to achieve network and cascade representation learning (NRL and CRL, respectively). Towards this end we follow two main directions: on one hand, we consider scenarios where the network structure is partially or completely known; on the other hand, we consider scenarios where the network structure is hidden. In particular, we give more emphasis to the latter case.

While the entire study could be motivated by the list of applications presented in Section 1.3, we first highlight why we need techniques based on diffusion events when the network structure is partially or fully hidden.

1.1 Motivation

Existing studies dedicated to both NRL and CRL are heavily reliant on the network structure. This is, however, a problem as there are several cases where one might lack the complete or partial structure of the network.

For example, consider online social networks (OSN) such as Twitter and Facebook. The follower/friend links on these networks are usually very difficult to obtain by interested third parties (researchers or businesses) working outside the OSN host companies, due to privacy policies and competitive market advantages [2]. Even when the data is publicly available, it may take several months to extract just a portion of the network. Another scenario could be an epidemic, where we know *who* has been infected and *when*, but not *how* they got infected.

Motivating example: Let us consider an excerpt of a hypothetical friendship subgraph of a Facebook like social network depicted in Fig. 1.1. Users of such social networks normally have the possibility to make their friendship links private, and hence let us suppose 2, 3, and 9 have done so.

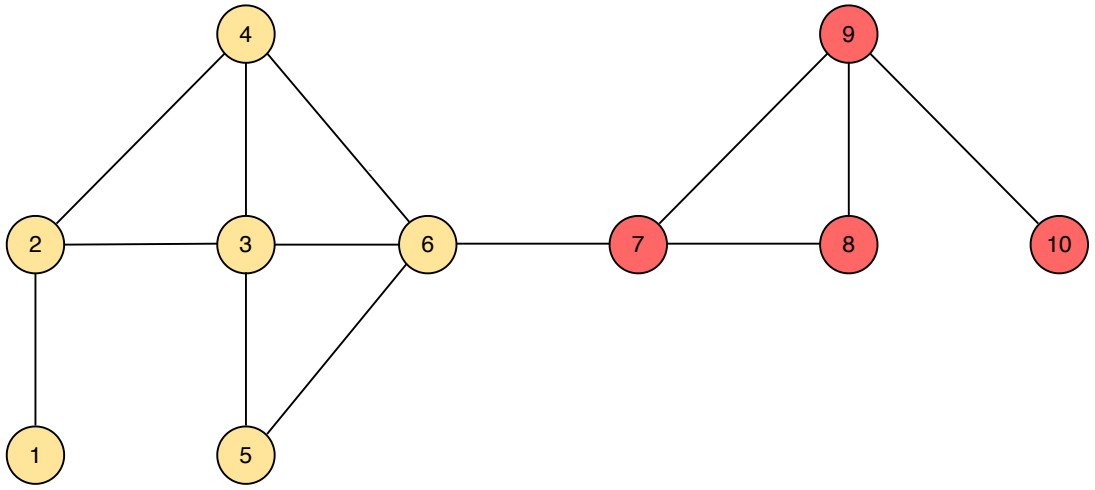


Figure 1.1: An example social network

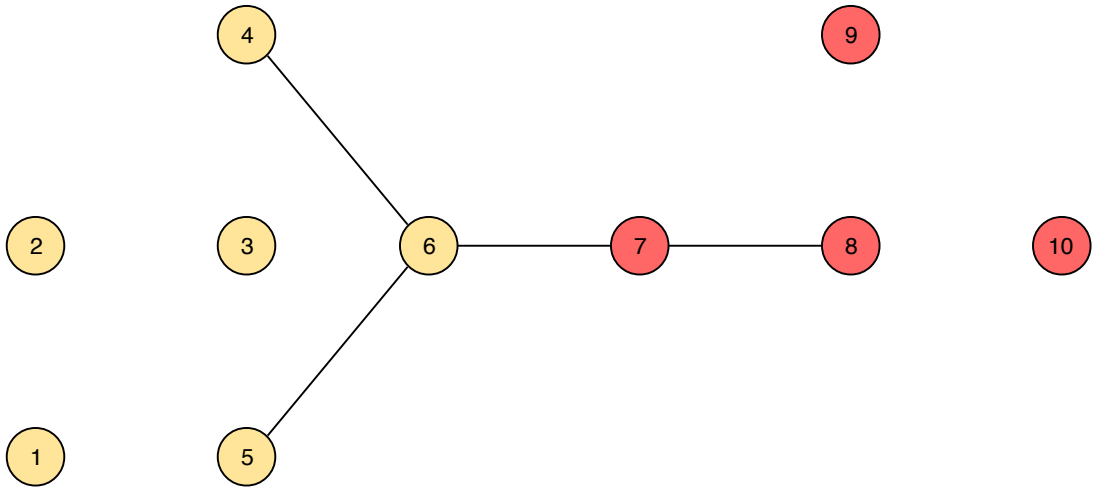


Figure 1.2: A subgraph of the social network in Fig. 1.1.

Now, consider a particular startup company xyz plc that wants to use the friendship network of the above social network for some of its services. A developer from xyz plc crawls the friendship network; the best she can manage to obtain, however, is the graph that is shown on Fig. 1.2, due to the privacy policy. Any representation R learned over such graph is apparently far from what one desires to achieve. We need to carefully examine other sources of information that might enable us to achieve a better

representation than R.

Fortunately, previous studies have shown a strong correlation between the dynamics of information diffusion events and the network structure [82, 83]. That is, in most diffusion events, similar users have the tendency to participate together. For example, if two users are connected, then they have a better likelihood of being involved in similar diffusion events with respect to a random user that is not connected to them. Therefore, one can use diffusion events as a proxy to the network structure by carefully probing the aforementioned kinds of correlations.

As a result, it is imperative to design representation learning algorithms – for both network and cascade prediction scenarios – that are not dependent on the knowledge of the complete network structure, but are still capable of producing "very good" results using information diffusion events.

1.2 Research challenges and contributions

We investigate the problem of representation learning in the context of informations graphs from three points of view, as discussed below.

Network representation learning with Structural Information

Similarly to most of the existing studies, the first contribution is to study the NRL problem when the network structure is already known. Our study is not limited to that, however; we consider also information coming from user interactions and other available attributes. The goal is to address the research question "*can we improve existing NRL results by incorporating the interactions among users and potential attributes describing them?*".

A naive solution can be provided by independently learning representations from three sources, i.e. the topology, the interactions and the attributes; then, these representation are concatenated into a single one.

This approach, however, fails to account for the correlation between the learned features. Thus, our main challenge here is to find an algorithm that learns a joint representation from three sources.

Towards this end, we first propose a simple edge-weighting scheme based on the attribute similarity of the endpoints of an edge. Second, we introduce a complementary approach to capture nodes proximity based on artificial or simulated information diffusion over a weighted graph. Finally, we incorporate actual or observed interaction histories that have led to diffusion events and combine them with the simulated diffusion events to complete the NRL task.

Unlike most existing techniques, the approach proposed here is robust to partially hidden network topologies, as it makes use of information diffusion events that occur over them.

We evaluate the effectiveness of our approach by comparing it over four real-world datasets against existing state-of-the-art techniques that are based on network structure only.

List of Publications

1. Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. Mineral: Multi-modal network representation learning. In Proc. of the 3rd International Conference on Machine Learning, Optimization and Big Data, MOD'17. ACM, September 2017.

2. Nasrullah Sheikh, Zekarias Kefato, and Alberto Montresor. GAT2VEC: Representation learning for attributed graphs. Computing, 2018.

3. Nasrullah Sheikh, Zekarias T. Kefato, and Alberto Montresor. HETNET2VEC: Semi-Supervised Heterogeneous Information Network Embedding for Node Classification using 1D-CNN. In Proc. of the First International Workshop on Deep and Transfer Learning (DTL'18).

IEEE, November 2018.

Network Representation Learning w/o Structural Information

Although scenarios where the information about the network structure is completely missing are particularly common, especially when dealing with social networks, only little emphasis has been given to the NRL problem in this context. Here we intend to address a second research question, that is, "*Can we learn an effective representation of nodes of a particular network when the structure is hidden?*"

Unlike the first contribution, however, here there is no straightforward way to see who is connected to who. Usually the NRL methods in the above category exploit the local neighborhood of nodes or the proximity between them. The lack of knowledge about the network structure constitutes a difficult challenge. That is, we need to find a mechanism where we can somehow compute the local neighborhood or the proximity of nodes in the original network by merely analyzing diffusion events only.

To address this problem, we propose different techniques to extract features from diffusion events that are indirectly related to the property of the original network. We achieve this by capitalizing on findings from previous studies regarding how the network structure and diffusion events over the network are related. Although it is inherently difficult to achieve performances as good as those obtained from NRL techniques based on a known structure, we have managed to obtain "reasonably close" results with respect to the state-of-the-art.

The ideas introduced to answer this research question are validated through an extensive experimental evaluation against real-world datasets and state-of-the-art NRL techniques that take advantage of the network structure.

List of Publications

1. Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. NetTensor: Network Representation Learning with Incomplete Information. Journal of Machine Learning Research, JMLR'18. Submitted for publication, September 2018.

2. Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. REFINE: Representation learning from diffusion events. In Proc. of the 4th Conference on Machine Learning, Optimization and Data science, LOD'18. Springer, September 2018.

3. Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. DeepInfer: Diffusion network inference through representation learning. In Proc. of the 13th International Workshop on Mining and Learning With Graphs, MLG'17. ACM, August 2017.

Representation Learning for Cascade Prediction

Finally, we examine how we can learn a representation of diffusion events (cascades) to predict their future states. Existing techniques for cascade prediction are mainly based on manual feature extraction. Such techniques, however, cannot be applied when the network structure is hidden. Furthermore, manual feature extraction could be expensive and time-consuming as it might depend on domain experts and external factors.

The third contribution answers thus the following research question: "*Can we learn representation of cascades that effectively predict their future states without the network structure and manually crafted features?*".

The challenge is twofold: we need to avoid manual efforts provided by domain experts and we should only depend on the information available in the cascades.

To meet this need, we propose a network-agnostic technique that automatically extracts features from cascades capable of effectively predicting them.

Similar to the previous contributions, we have also experimentally evaluated the performance of this algorithm using data from two popular social networks and compared it against the state-of-the-art.

List of Publications

1. Zekarias T. Kefato, Nasrullah Sheikh, Leila Bahri, Amira Soliman, Alberto Montresor and Sarunas Girdzijauskas. CAS2VEC: Network Agnostic Cascade Prediction in Online Social Networks. In Proc. of the 5th Conference on Social Network Analysis, Management and Security, SNAMS18. IEEE, November 2018. (**Best Paper**)

1.3 Applications

Traditionally, to analyze information networks, one has to engineer separate sets of features for different kinds of applications. Nonetheless, NRL techniques turn out to be incredibly useful as they spare us from the repeated feature engineering and empower us to use the same representation across different applications.

In this section, we discuss some of the major application areas of both network and cascade representation learning when the same set of representations can be used.

Network Reconstruction

An important problem in different fields such as systems biology, epidemiology and social sciences is the reconstruction of a hidden interaction network of agents based on observed traces of diffusion events. As an example,

consider the reconstruction of the underlying interaction network between people infected by an epidemic.

One of the constraints of NRL is that the representation should preserve the most important properties of the underlying network, such as the structure and topology of the network. If the structure of the network is preserved while embedding the nodes of the graph into a latent vector space, then one should be able to reconstruct the original network.

An interesting property of diffusion events is that they unveil an interesting pattern that describes how the agents are linked or interact in the underlying hidden network. That is, nodes participation in diffusion events follows a certain pattern that indicate their structural connection.

Therefore, one can embed the agents participating in diffusion events into a latent vector space so as to capture the aforementioned patterns. This is indirectly equivalent to preserving the structure of the network as the patterns in cascades are related to the network structure. Then, one can reconstruct the network, for instance by computing distance or similarity between the embedding vectors of the agents.

Link Prediction

Link prediction is one of the most important problems in social network analysis. The task is to recommend possible connections between nodes that are currently not connected but are highly likely to be connected in the future. In general, there are three main directions towards solving this problem, which are based on node similarity, topology, and social theory [79]. Very often, such techniques rely on experts to craft informative features that tries to capture nodes local neighborhood information.

On the other hand, learned embeddings are usually designed in such a way that they capture the local neighborhood information of nodes. This property of NRL algorithms naturally benefits link prediction algorithms.

For example, in a traditional link prediction pipeline, one can use node embeddings learned through a NRL algorithm instead of manually crafted features.

Node Classification

In node classification, the goal is to assign labels to unlabeled nodes based on the labels associated to their neighbors. Node classification algorithms are usually designed based on the theory of homophily from social studies, which argues that nodes with similar characteristics have the tendency to be connected.

Akin to link prediction, traditionally, features that capture the similarity between the properties of nodes are manually crafted by experts. Yet again, features can be learned using NRL algorithms and node classification can be carried out using node embeddings.

Network Visualization

Before designing a particular algorithm for a given problem, the first step is to understand the data at hand. Visualization is one of the most important tools to acquire insights about the data. For example, it enables to understand whether the graph is organized into modular components or communities.

Visualizing the community structure of a small graph might be easy; as the size of graph grows, however, obtaining a visualization where one can draw meaningful insights is challenging.

To solve this problem, it is useful to project the graph into a lower-dimensional vector space and build a representation based on such space. It is easy to see how NRL techniques fit in such application.

Rumor Detection

After having seen several applications of NRL techniques, we conclude by providing an example related to cascade prediction.

In *rumor detection*, we are usually interested in identifying online contents that are intended to misinform or spread false/fake news to social network users. Classifying whether a certain content is rumor, however, is to no avail if it is not going to affect a "significant" number of users, or if the effect of the content is not "significant". Thus one has to first identify whether a post has a "significant" effect or not before a rumor detection task. We can also turn the problem upside down and decide whether a content classified as a rumor is going to have a "significant" effect.

Regardless of the choice of the order of the rumor detection task, it is important to identify the effect of a content. Usually this task is known as popularity or cascade prediction. Therefore one can use the representation learning for cascade prediction component to aid the rumor detection task.

1.4 Structure of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2 we introduce the formal graph and cascade models and other definitions that are used throughout the paper. Next, in Chapter 3 we give a comprehensive background for all the algorithms used in the thesis. From Chapter 4 to Chapter 6 we discuss the three algorithms we propose to address the three research questions raised in Section 1.2. The state-of-the-art is covered in Chapter 7 and we conclude the thesis in Chapter 8.

Chapter 2

Models and Preliminaries

This thesis is focused on the concepts of information networks and cascades (a.k.a. information diffusion events). This chapter introduces the notation used in the rest of the work, including the basic concepts that model information networks and the diffusion of information across them, also know as cascades.

2.1 Information Network

Throughout the thesis, we consider generic information networks that can be described as directed, weighted, and labeled graphs $G = (V, E, \mathbf{W}, a)$, where V is a set containing n nodes, $E \subseteq V \times V$ is a set containing m edges, $\mathbf{W} \in \mathbb{R}^{n \times n}$ is a weight matrix of size $n \times n$, and $a : V \rightarrow \mathcal{A}$ that associates each node $u \in V$ with a set of attributes $a(u)$ taken from an attribute collection $\mathcal{A}$. $\mathbf{W}[i, j]$ is the entry in the i^{th} row and j^{th} column of $\mathbf{W}$ and corresponds to the weight associated to the edge $(i, j) \in E$. We use Eq. 2.1 and 2.2 to denote the set of outgoing and incoming neighbors of node i, respectively.

$$out(i) = \{j : (i, j) \in E\} \tag{2.1}$$

$$in(i) = \{j : (j, i) \in E\} \tag{2.2}$$

Note that the proposed techniques can be easily applied to undirected and unweighted graphs as well, by simply putting two directed edges (i, j) and (j, i) for every undirected edge $(i, j) \in E$ and by assigning a constant weight $w(i, j) = 1$ to all edges.

2.2 Information Diffusion Events

Usually, nodes in information networks are involved in activities such as content generation, message exchanges, information sharing and so on. In this study, we focus on information sharing activities that lead to the spread or diffusion of a piece of content. We refer to the piece of content as a *contagion*; it can be a piece of text, a picture, a video or any valid content that can be generated by nodes in information networks.

The first node that generates a contagion is called the *origin*; the last node where the diffusion of the contagion stops is called the *sink*. We refer to the moment when a piece of content reaches a node $v \in V$ as the *infection event* of node v. The complete propagation process of a contagion from the origin up to the sink is called a *diffusion event* or *cascade*.

More formally, a cascade $C \in \mathcal{C}$ is defined as an ordered sequence of infection events:

$$C = [(u_1, t_1), \ldots, (u_{|C|}, t_{|C|})], \tag{2.3}$$

where (u_i, t_i) is the i^{th} infection event and u_i and t_i are the i^{th} infected node and infection timestamp, respectively. We denote the set of all cascades as $\mathcal{C} = \{C_1, \ldots, C_c\}$.

Note that indexing in cascades does not pertain to node identities; it rather refers merely to the order of infection events; that is, here i is only associated to a position in a cascade. As infection events are ordered, such that $1 \leq i < j \leq |C| \Rightarrow t_i < t_j$, for any cascade C and any pair of indices

i, j.

In addition, for any cascade C, the timestamp associated to the first infection event t_1 is always 0, and the following timestamps are relative to it:

$$C = [(u_1, t_1 = 0), \ldots, (u_{|C|}, t_{|C|})]$$

We also assume that nodes can only be infected once, therefore in a cascade C if a node $u \in V$ is infected at infection time step i, that is,

$$C = [\ldots, (u_i = u, t_i), \ldots]$$

then there is no index $j \neq i$ such that

$$C = [\ldots, (u_j = u, t_j), \ldots]$$

2.3 Additional notations

We use $C(i) = (u_i, t_i)$ to denote the i^{th} event of a cascade C. To easily identify the set of all cascades associated with a particular node $v \in V$, we use $\mathcal{C}_v$ defined as:

$$\mathcal{C}_v = \{C : \exists j \wedge 1 \leq j \leq |C| \wedge C(j) = (u_j = v, t_j)\} \tag{2.4}$$

We consider a generic *similarity function* $sim : V \times V \times \mathbb{R}^{x \times y} \rightarrow [0, 1]$, that measures the similarity between pairs of nodes u, v as described by the row vectors $\mathbf{W}[i]$ and $\mathbf{W}[j]$, respectively. For example, we could adopt cosine similarity:

$$sim(i, j; \mathbf{W}) = cos(\mathbf{W}[i], \mathbf{W}[j])$$

The aforementioned formulation captures similarity between nodes merely based on their position in the network. However, that is a simplified assumption; in reality, a richer set of factors play crucial roles in deciding

Sample Symbol	Description
$\mathbf{X}$, Ψ	A convention used to denote a matrix – bold caps letter or Greek caps letter
$\mathbf{X[i]}$ or $\mathbf{x_i}$	A convention used to denote the i^{th} row of the matrix $\mathbf{X}$
$\mathbf{X[i, j]}$ or $\mathbf{x_i[j]}$	A convention used to denote the i^{th} row and j^{th} column of the matrix $\mathbf{X}$
$\mathbf{x}$	A convention used to denote a vector – bold small letter
$\mathbf{x[i]}$	A convention used to denote the i^{th} component of the vector $\mathbf{x}$
$\mathbf{X}^T$, $\mathbf{x}^T$	A convention used to denote the transpose of a matrix or a vector, respectively
S	A convention used for sequence or set of scalar values is denoted by caps letter
$\mathcal{S}$	A set of sets or sequences is denoted by a calligraphic symbol
G	An information graph
V	The set of nodes of G
E	The set of edges of G
$\mathbf{W}$	The weight matrix associated with G
n	The number of nodes
m	The number of edges
$\mathcal{C}$	The set of cascades
$\mathcal{C}_i$	The set of cascades of node i
Φ	A low-dimensional dense embedding or representation matrix
$\mathbf{H^k}$	The weight-matrix of the k^{th} hidden layer of a feed-forward neural network
$in(u)$	Incoming neighbors of a node $u \in V$
$out(u)$	Outgoing neighbors of a node $u \in V$
$N(u) = in(u) \cup out(u)$	Neighbors of a node $u \in V$

Table 2.1: Notations and Conventions

how similar nodes are [56]. In this study we enrich the similarity function by incorporating node attributes and/or activity or interaction profiles.

For ease of discussion, let us consider an overloaded oracle SIM that

computes similarity between a pair of nodes based on any given input space. The input space could be topological information ($\mathbf{W}$ – Eq. 2.5), attribute information ($\mathcal{A}$ – Eq 2.6), interaction profiles recorded in cascades ($\mathcal{C}$ – Eq. 2.7), or the combination of one or more of them (Eq. 2.8).

$$\text{SIM} : V \times V \times \mathbf{W} \rightarrow [0, 1] \tag{2.5}$$

$$\text{SIM} : V \times V \times \mathcal{A} \rightarrow [0, 1] \tag{2.6}$$

$$\text{SIM} : V \times V \times \mathcal{C} \rightarrow [0, 1] \tag{2.7}$$

$$\text{SIM} : V \times V \times \mathbf{W}^* \times \mathcal{A}^* \times \mathcal{C}^* \rightarrow [0, 1] \tag{2.8}$$

In Eq. 2.8, the $*$ indicates that the corresponding input space is optional. Hence, it is not necessary for all the input sources to be specified; SIM can be defined depending on the availability of the input spaces. For example, using just the cascade input space, SIM could be computed as the probability of node u to occur in a cascade, under the condition that v has also occurred (in any order) (Eq. 2.7).

$$\text{SIM}(u, v; \mathcal{C}) = p(u|v) \tag{2.9}$$

The conditional probability $p(u|v)$ could be estimated using a simple normalized co-occurrences between the pairs as:

$$p(u|v) = \frac{|\mathcal{C}_u \cap \mathcal{C}_v|}{|\mathcal{C}|} \tag{2.10}$$

We can also further improve the simple estimation of $p(u|v)$ by introducing strong constraints like co-occurrences within a context window [63] and so on.

Finally, throughout the thesis we use terms *embedding* and *representation* of nodes and cascades interchangeably. The list of notations and conventions used throughout the thesis are presented in Table 2.1.

Chapter 3

Background

In order to solve the three research questions discussed in Chapter 1, we have developed algorithms that are based on already existing matrix factorization and neural network models. In the former models, we adopt both traditional matrix factorization and more recent approaches based on neural networks. The main motivation for using neural matrix factorization (NMF) is their capability to model non-linear problems. As the structure and dynamics of diffusion events over information networks are highly non-linear, NMF techniques are the perfect fit. In the latter models, we adopt both shallow and deep neural networks.

For the sake of completeness, in this chapter we give a detailed discussions of the algorithms that we have adopted in the thesis. The following is the list of algorithms employed:

1. Truncated Singular Value Decomposition
2. Neural Non-Negative Matrix Factorization
3. SkipGram
4. AutoEncoder
5. Convolutional Neural Networks

3.1 Truncated Singular Value Decomposition (TSVD)

SVD is a matrix factorization technique which decomposes a given matrix $\mathbf{Z} \in \mathbb{R}^{N \times M}$ with rank $K \leq min(M, N)$ as follows:

$$\mathbf{Z} = \mathbf{A}\Sigma\mathbf{B}^T = \sum_i \sigma_i \cdot \mathbf{a}_i \cdot \mathbf{b}_i^T \tag{3.1}$$

where $\mathbf{A} \in \mathbb{R}^{N \times K}$, $\mathbf{B}^T \in \mathbb{R}^{K \times N}$ are column orthonormal matrices – $\mathbf{A}\mathbf{A}^T = \mathbf{B}\mathbf{B}^T = \mathbf{I}$, and $\Sigma = diag(\sigma_1, \ldots, \sigma_K)$ are singular values. Computing the full-rank SVD factorization is expensive for large matrices, and not relevant to our problem. Therefore we use the truncated generalized version (TSVD) [33] that has been shown to be scalable for large-scale graphs [59]. Here, instead of the full-rank K decomposition, TSVD computes the basis of a k–dimensional left singular subspace of $\mathbf{Z}$, where $k \ll K$, using for example power iteration techniques [4].

If $\mathbf{Z}$ is not symmetric ($\mathbf{Z}[\mathbf{i}] \neq \mathbf{Z}[\mathbf{i}]^T$), then following [59] we decompose it into two factor matrices $\mathbf{X} \in \mathbb{R}^k$ and $\mathbf{Y} \in \mathbb{R}^k$ as

$$\mathbf{X} = [\sqrt{\sigma_1} \cdot \mathbf{a}_1, \ldots, \sqrt{\sigma_k} \cdot \mathbf{a}_k] \tag{3.2}$$

$$\mathbf{Y} = [\sqrt{\sigma_1} \cdot \mathbf{b}_1, \ldots, \sqrt{\sigma_k} \cdot \mathbf{b}_k] \tag{3.3}$$

Otherwise, if $\mathbf{Z}[\mathbf{i}] = \mathbf{Z}[\mathbf{i}]^T$, we only materialize $\mathbf{X}$.

3.2 Neural Non-Negative Matrix Factorization (NNMF)

Given a non-negative matrix $\mathbf{Z}$, the goal of non-negative matrix factorization is to find two lower-rank factor matrices $\mathbf{X}$ and $\mathbf{Y}$ that approximate $\mathbf{Z} \approx \mathbf{X}\mathbf{Y}$ provided the constraints $\mathbf{X} \geq 0$ and $\mathbf{Y} \geq 0$ are satisfied. To quantify the quality of the approximation, different optimization strategies could be employed; for example, both distance- (such as Euclidean)

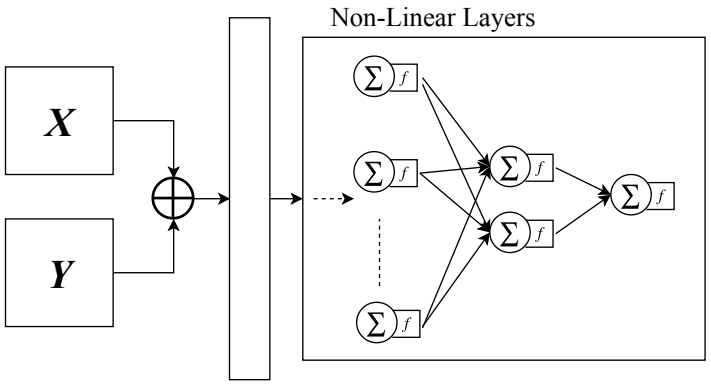


Figure 3.1: Architecture of the neural-non-negative matrix factorization model. Dotted lines indicate optional components. If the dotted box is left out, the NNMF is equivalent to NMF and instead of concatenation we compute dot product.

and divergence-based approaches (such as Kulback, Bregman) [50, 16] have been used. For example, using the Euclidean distance, the objective is specified as

$$\min ||\mathbf{Z} - \mathbf{XY}||_F^2 = \sum (\mathbf{Z}[\mathbf{i},\mathbf{j}] - (\mathbf{X}[\mathbf{i}] \cdot \mathbf{Y}[\mathbf{j}]^T))^2 \qquad (3.4)$$
$$subj.\ to \quad \mathbf{X}, \mathbf{Y} \geq 0$$

$\mathbf{X}$ and $\mathbf{Y}$ are then iteratively updated until convergence using algorithms such as stochastic gradient descent (SGD) or its variants. Due to its expressive power that enables us to incorporate non-linearity, in this work we propose a simple and general neural-non-negative matrix factorization (NNMF) for extracting latent factors. The general overview of the model is given in Fig. 3.1 and we adjust the optimization objective in Eq. 3.4 as follows

$$\min ||\mathbf{Z} - \texttt{layers}(\mathbf{H^1} \cdot (\mathbf{X} \oplus \mathbf{Y}))||_F^2 = \sum (\mathbf{z_i}[\mathbf{j}] - \texttt{layers}(\mathbf{H^1} \cdot (\mathbf{x_i} \oplus \mathbf{y_j}))^2$$
$$subj.\ to \quad \mathbf{X}, \mathbf{Y} \geq 0 \qquad (3.5)$$

where `layers` is a composition of non-linear functions of linear trans-

formations ('Non-Linear Layers' in Fig. 3.1), and $\mathbf{H^i}$ is the weight matrix of the i^{th} hidden layer. For all the layers except the last one, we use tanh; we use $relu$ at the output layer instead, to predict the non-negative entries of $\mathbf{Z}$. The input is specified by the concatenation of $\mathbf{X}$ and $\mathbf{Y}$ using the $\oplus$ operation as shown in Fig. 3.1.

The task is to predict the entries $\mathbf{z_i[j]}$ of $\mathbf{Z}$ using the current projections of $\mathbf{x_i}$ and $\mathbf{y_j}$. In every iteration, a prediction $\tilde{\mathbf{z}}_\mathbf{i}[\mathbf{j}]$ of $\mathbf{z_i[j]}$ is computed as

$$\begin{aligned}\tilde{\mathbf{z}}_\mathbf{i}[\mathbf{j}] &= \texttt{layers}(\mathbf{H}^1 \cdot (\mathbf{x_i} \oplus \mathbf{y_j})) \\ &= \texttt{relu}(\mathbf{H}^L \cdot \texttt{tanh}^{L-1}(\ldots \mathbf{H}^2 \cdot \texttt{tanh}^1(\mathbf{H}^1 \cdot (\mathbf{x_i} \oplus \mathbf{y_j}))\ldots))\end{aligned} \tag{3.6}$$

where L is the number of layers and $\mathbf{H}^L$ is the weight matrix of the L^{th} layer. Note that $\mathbf{H}^L$ is a vector as we have only one neuron at the output of the model and $\mathbf{H}^L$ has the same number of components as the output of the (L-1)th layer.

Finally, depending on the incurred loss (Eq. 3.5) of each of the iterations, $\mathbf{X}$ and $\mathbf{Y}$ will be updated until convergence or the squared distance (loss) is minimized ($||\mathbf{Z} - \tilde{\mathbf{Z}}||_F^2 \approx 0$), using stochastic gradient descent.

3.3 SkipGram

The SkipGram [57] model has been developed inside the natural language processing community. It has been used for language modeling based on the distributional hypothesis from linguistics. According to this hypothesis, words meaning can be understood by a proper examination of their context [20, 21]. That is, a word cannot have a meaning without other words frequently accompanying it in its occurrence, and this is the notion that was introduced by linguists like J. R. Firth (known for his famous quote "you shall know a word by the company it keeps").

The essential idea of the SkipGram model, depicted in Fig. 3.2, is

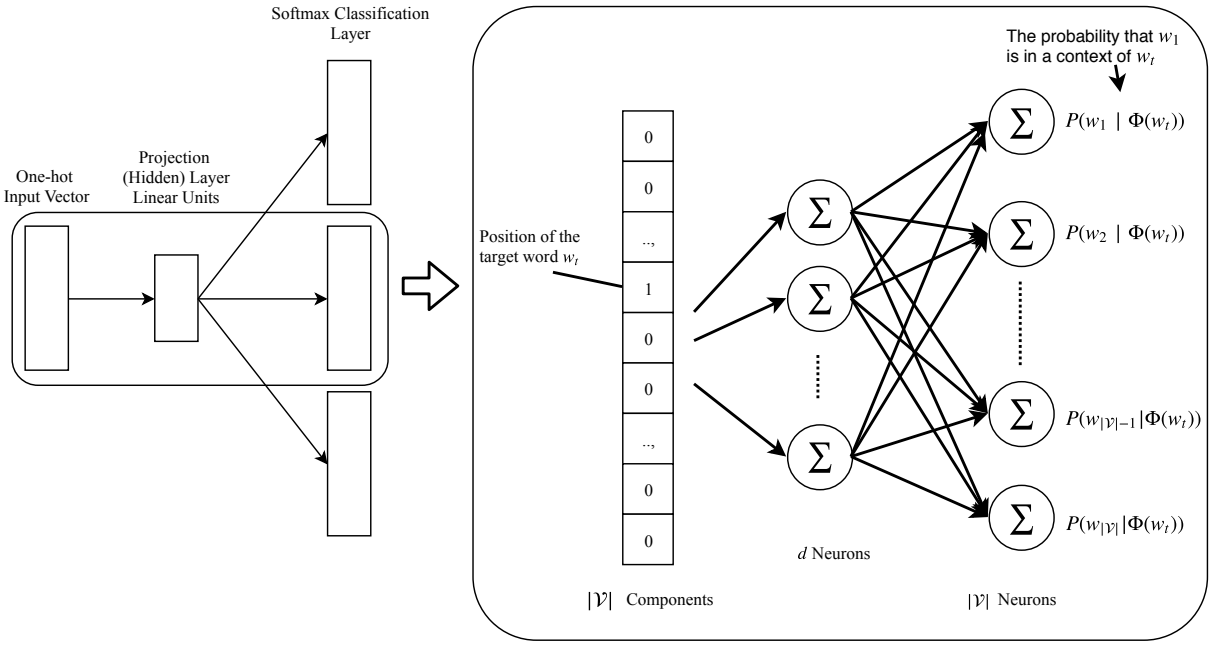


Figure 3.2: Architecture of the SkipGram model

to project words into a latent continuous vector space that preserves the distributional semantics of words. That is, in the vector space, words that tend to co-occur within a context should be projected close to each other. In other words, we seek to learn a low-dimensional representation (the 'Projection Layer' of Fig. 3.2) of a target word that is capable of predicting its nearby or context words. To formally define the model, suppose we have a document corpus $\mathcal{D} = \{D_i : 1 \leq i \leq |\mathcal{D}|\}$, and let $\mathcal{S}_i$ be the set of sentences in document D_i, and $S \in \mathcal{S}_i$ be a sentence specified as a sequence of words:

$$S = w_1, w_2, \ldots, w_{|S|}, \tag{3.7}$$

where each word w_i is a sample from a vocabulary $\mathcal{V}$. Consider the functions $S(i) = w_i$ that returns the i^{th} word in the sentence S and function $context(S, w_t, s)$ that extracts the context words of w_t from the sentence S

within a context size s:

$$context(S, w_t, s) = \{S(j) = w_j : 1 \leq t - s \leq j \leq t + s \leq |S|, j \neq t\}$$

Then, for each target word $S(t) = w_t$ and a given context size s, the SkipGram model maximizes the log probability of observing the set of context words given the current target word:

$$\max \sum_{t=1}^{|S|} \log P(context(S, w_t, s)|w_t) \tag{3.8}$$

$$\max \sum_{t=1}^{|S|} \sum_{w_c \in context(S, w_t, s)} \log P(w_c|w_t) \tag{3.9}$$

As the goal is to embed words into a latent vector space, we condition on the current embedding of the target word $\Phi(w_t)$ and rewrite Eq. 3.9 as

$$\min -\sum_{t=1}^{|S|} \sum_{w_c \in context(S, w_t, s)} \log P(w_c|\Phi(w_t)) \tag{3.10}$$

The technique employed by the SkipGram model is also known as "*learning by prediction*" [3]. This is due to the optimization objective in Eq. 3.10 that is formulated as a prediction task of each context word given the embedding of a target word. The prediction task is normally handled as a softmax classification problem (hence the "Softmax Classification Layer" of Fig. 3.2) :

$$P(w_c|\Phi(w_t)) = \frac{exp(\Phi(w_c)^T \cdot \Phi(w_t))}{\sum_{w \in \mathcal{V}} exp(\Phi(w)^T \cdot \Phi(w_t))} \tag{3.11}$$

That is, the model produces a conditional distribution $P(w_i|\Phi(w_t))$ that encodes the probability that each word $w_i \in \mathcal{V}$ is a context word of a target word w_t. Next, the prediction $P(\cdot|\Phi(w_t)) \in [0, 1]^{|\mathcal{V}|}$ is checked against

each encoding $\mathbb{1}_{w_c}$ of the ground truth context word $w_c \in \mathit{context}(S, w_t, s)$, where $\mathbb{1}_{w_c} \in \{0,1\}^{\mathcal{V}}$ is a one-hot vector of w_c that has 1 for the component associated with the position of w_c and 0 everywhere as shown below.

$$\mathbb{1}_{w_c} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

However, training the SkipGram model using the softmax formulation in Eq. 3.11 is very expensive due to the normalization constant that needs to be computed over all the vocabulary words $w \in \mathcal{V}$. In this thesis we adopt the "negative sampling" technique [57] that characterizes a good model by its power to discriminate noise from the correct context words. Then, the computation of $\log P(w_c|\Phi(w_t))$ using the negative sampling strategy is specified as:

$$P(w_c|\Phi(w_t)) = \log \sigma(\Phi(w_c), \Phi(w_t)) + neg(w_t, \ell), \tag{3.12}$$

where σ is a sigmoid function, and the goal is to train a model that is capable of effectively differentiating the proper context word w_c of w_t from the ℓ negative (noisy) samples w_n drawn from some noise distribution $\mathcal{N}(w_t)$ of the target word w_t. $neg(w_t, \ell)$ is the noise model, defined as:

$$neg(w_t, \ell) = \ell \cdot \mathbb{E}_{w_n \sim \mathcal{N}(w_t)}[-\log \sigma(\Phi(w_n), \Phi(w_t))] \tag{3.13}$$

Equation 3.12 is equivalent to querying the model regarding its belief that w_c is a correct context word of w_t as opposed to the negative samples w_n. Numerically, a good model should produce a small expected probability for the noise model and larger probability for the data model (the first term on the right-hand-side of Eq. 3.12).

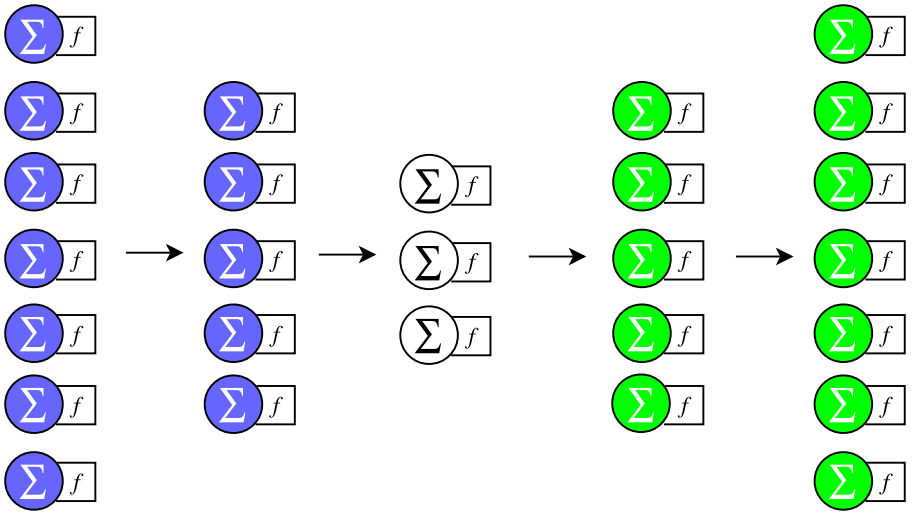


Figure 3.3: A standard architecture of an autoencoder. Each units of the feed-forward network constitutes a non-linear activation (f) of the linear transformation $\sum$ of its input.

Finally, one can train the model using the entire document corpus $\mathcal{D}$ and an optimization algorithm like stochastic gradient descent (SGD). After a proper training – or after the model parameters (Φ) are tuned – we can use the embedding $\Phi(w)$ of each word $w \in \mathcal{V}$ for different kinds of natural language processing tasks.

3.4 AutoEncoder

An *autoencoder* (Fig. 3.3) is a feed-forward neural network that is commonly used for an unsupervised machine learning. It has two components, which are called *encoder* and *decoder*. The encoder (the blue layers) is used to generate a compact and dense representation (the white layer) of a typically sparse and high-dimensional input data. The decoder (the green layers) tries to reconstruct the original input data from the compressed code. Informally, the objective is to train a model that learns a high-

quality compact representation of the input data capable to reconstruct the input data from the representation.

More formally, let $\mathbf{I}$ be the high-dimensional input data to the autoencoder. Then we define, respectively, the encoder ENC and DEC in Eq. 3.14 and 3.15 as a composition of non-linear functions

$$\Phi = \text{ENC}^{L}(\mathbf{H}^{L}_{\mathbf{enc}} \cdot \text{ENC}^{L-1}(\dots \mathbf{H}^{\mathbf{2}}_{\mathbf{enc}} \cdot \text{ENC}^{\mathbf{1}}(\mathbf{H}^{\mathbf{1}}_{\mathbf{enc}} \cdot \mathbf{I}) \dots)) \tag{3.14}$$

$$\mathbf{I}' = \text{DEC}^{L}(\mathbf{H}^{L}_{\mathbf{dec}} \cdot \text{DEC}^{L-1}(\dots \mathbf{H}^{\mathbf{2}}_{\mathbf{dec}} \cdot \text{DEC}^{\mathbf{1}}(\mathbf{H}^{\mathbf{1}}_{\mathbf{dec}} \cdot \Phi) \dots)) \tag{3.15}$$

where L is the number of layers, $\mathbf{H}^{\mathbf{l}}_{\mathbf{enc}}$ and $\mathbf{H}^{\mathbf{l}}_{\mathbf{dec}}$ are the weight matrices, and $\text{ENC}^{\mathbf{l}}$ and $\text{DEC}^{\mathbf{l}}$ are the non-linear functions, such as `sigmoid` and `tanh`, of the l^{th} layer of the encoder and decoder, respectively.

The training objective is commonly modeled as a minimization of the input reconstruction error

$$\min ||\mathbf{I} - \mathbf{I}'||^2_F \tag{3.16}$$

Very often, due to the sparsity of $\mathbf{I}$, the basic formulation is prone to learning to reconstruct the zeros, and a simple penalization trick is commonly used to avoid this [77]. For this reason, the model is strongly penalized when it fails to reconstruct the non-zero elements and weakly penalized for the zeros of the input data. This trick is achieved by introducing a term $\mathbf{S}$ to the above equation as follows

$$\min ||(\mathbf{I} - \mathbf{I}') \otimes \mathbf{S}||^2_F \tag{3.17}$$

where $\otimes$ is the Hadamard product and $\mathbf{S} \in \mathbb{R}^{n \times n}$ is associated with the input $\mathbf{I}$, *i.e.* if $\mathbf{I}[\mathbf{i}, \mathbf{j}] = 0$, then $\mathbf{S}[\mathbf{i}, \mathbf{j}] = 1$; otherwise $\mathbf{S}[\mathbf{i}, \mathbf{j}] = \mu > 1$.

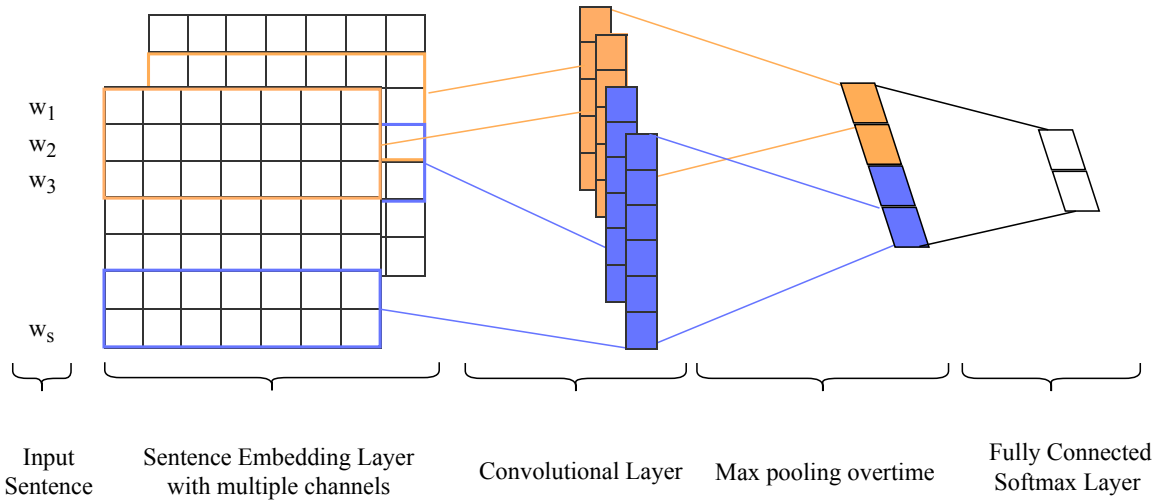


Figure 3.4: CNN model for sentence classification

The larger the value of μ, the stronger the penalty is on the reconstruction errors for the non-zero entries of $\mathbf{I}$.

Usually, a regularization term or a dropout regularization technique is used to avoid over-fitting. Finally, one can train the model by optimizing either of the objectives in Eq. 3.16 or 3.17 depending on the sparsity of the input data $\mathbf{I}$ using algorithms such as stochastic gradient descent (SGD).

3.5 Convolutional Neural Networks (CNN)

Convolutional neural networks are widely used in different areas, such computer vision and NLP. In particular, they have proved to be highly effective in tasks such as object detection/recognition.

Generally, we can group them into two broad categories based on the type of input they process: 2-dimensional objects (such as images, video frames) and 1-dimensional objects (such as time-series, signals, and sentences in documents). In this thesis, we focus on the 1D variant that we have adapted to our problem.

In the following, we give an overview of the model introduced for sen-

tence classification [42], whose architecture is depicted in Fig. 3.4.

The model takes a sentence $S = w_1, \ldots, w_s$ of length s as input. Each sentence is embedded using a multi-channel embedding matrix. In the approach proposed by Kim [42], a fixed channel, based on pre-trained word embeddings and trainable channel that can be optimized with respect to the task-at-hand, for example sentence classification, have been used.

We illustrate now how a classification is carried out during inference time, supposing that all the model parameters have been already trained. Given an input test sentence $S_t = w_1, \ldots, w_s$, its embedding matrix $\mathbf{E} = [\mathbf{we_1}, \ldots, \mathbf{we_s}]^T$ is constructed using the trained word embedding vectors $\mathbf{we_i} \in \mathbb{R}^d$ of each word $w_i : i = 1, \ldots, s$. This corresponds to the sentence embedding layer in Fig 3.4.

Then, in the convolutional layer the prediction task starts by applying a set of p filters on the cascade embedding matrix. That is, we apply p filters (denoted by different colors) of different sizes on every possible slice of the input of the convolutional layer (the cascade embedding matrix). More formally,

$$\phi_{i,r} = \sigma(\mathbf{h_i} \cdot \mathbf{e_{jk}} + b) \tag{3.18}$$

where the vector $\mathbf{h_i} \in \mathbb{R}^{kd}$ is the i^{th} filter (kernel), $b \in \mathbb{R}$ is the bias, σ is an activation function, such as `relu`, k is the size of the i^{th} filter, $\phi_{i,r}$ is the feature value of the i^{th} filter on the r^{th} round convolution, and $\mathbf{e_{jk}} = \mathbf{e_j} \oplus \ldots \oplus \mathbf{e_{j+k}} \in \mathbb{R}^{kd}$ is a concatenation of k rows of the matrix $\mathbf{E}$, starting from the j^{th} row. Generally, the i^{th} filter of size k is convolved $s-k+1$ times, to give the feature maps $\boldsymbol{\phi}_i = [\phi_{i,1}, \ldots, \phi_{i,s-k+1}]$. $\boldsymbol{\phi}_i$ captures patterns in high-level features, such as *n–grams* in language documents.

Next, a max-pooling (or a max-pooling overtime) operation is applied over each feature map to sample from a specified pooling window, and the simplest technique is the one that draws from the entire feature vector, *i.e.* $\max(\boldsymbol{\phi}_i) = \widehat{\phi}_i \in \mathbb{R}$. Intuitively, this corresponds to selecting the

best feature that is activated when a certain pattern in the input space is detected. The max-pooling output, more formally $\mathbf{z} = [\widehat{\phi_1}, \ldots, \widehat{\phi_p}]$, is followed by a fully connected softmax classification layer. The vector $\mathbf{z}$ can be viewed as the final set of features extracted for the current sentence, and it will be used to predict the sentence into one of the $l \geq 2$ target classes $\mathcal{Y} = \{y_i : 1 \leq i \leq l\}$. This model was original proposed for binary classification $l = 2$, henceforth we assume this setting.

The above description assumes that the model is trained; to perform the training, the optimization objective of the model is specified as the minimization of the misclassification error of the sentences. More formally, we adopt the standard binary cross-entropy objective function:

$$\min -\sum_i y_i \log(\texttt{model}(S_i)) + (1 - y_i) \log(1 - \texttt{model}(S_i)) \tag{3.19}$$

Here, S_i and $y_i \in \{0, 1\}$ are the i^{th} sentence and class label, respectively. `model` is the proposed model that produces a probability distribution (prediction) $\mathbf{y}$ for the given training sentence S_i over the classes (1 and 0):

$$\mathbf{y} = \mathbf{h} \cdot (\mathbf{z} \circ \mathbf{v}) + b \tag{3.20}$$

where $\mathbf{v} = [v_1, \ldots, v_p]$ is a Bernoulli distribution used for dropout regularization as proposed in [42], with $\mathbf{v_i} \in \{0, 1\}$.

Ultimately, the model parameters $[\mathbf{E}, b, \mathbf{h_i}, \mathbf{h}]$ are trained using the back-propagation algorithm.

Chapter 4

Network Representation Learning with Structural Information

As we have discussed in the introduction, we approach the NRL task from two perspectives. This chapter is dedicated to the first one, where the network structure could be partially or completely known.

In general, the goal of a NRL task is to embed the nodes of the graph into a latent vector space in such a way that the most important properties of the network are preserved. Among such properties, for instance, those associated to the topology of the network are particularly important. For example, connectivity of nodes, the shape of their neighborhood, or their organization as a community; in general, the global proximity that each node has with rest of the nodes in the network. Preserving these properties while embedding the nodes in the latent space enables one to perform different kinds of network analysis, such as link prediction and node classification, in an efficient and effective way.

Thus, in order to obtain high-quality node embeddings that are going to be successful in the aforementioned tasks, we need to carefully investigate factors that govern nodes proximity or that could be attributed to their observed proximity.

In other words, first we should examine what factors have led users to

be located in a close proximity towards a given set of users and farther apart from others in a given information network. One of the most widely accepted theories from social science regarding the mechanism by which nodes might end up in a close proximity or create connection between them is homophily [56]. According to homophily, nodes prefer to connect with other nodes with whom they share similar properties, creating tightly connected communities. For example, in online social networks such as Facebook, users are more likely to befriend others with whom they share common characteristics (field of study, religion, race, schools, interests, etc.) rather than a random and unknown user [56, 88]. In addition, the homophily theory also highlights that ties between non-homogeneous nodes have less chance of surviving the test of time.

For this purpose, similar to [88], we associate a functional essence to every node in an information network, that characterize its intrinsic preferences. The following can be taken as examples of functional aspects of nodes in an information network:

- profile of a user in a social network
- attributes of an author in a scientific collaboration network
- functions of a protein in a protein interaction network

Secondly, we need to account for observed proximities, through the factors that explain the observation itself, or through factors that could serve as a proxy for a certain level of observed proximity.

In summary, we account for three facets of an information network in learning embeddings of nodes:

- nodes functional essence
- observed proximity

- an indicator of the observed proximity

We assume that the nodes functional essence is the key factor that governs why nodes might prefer to be part of particular neighborhood in the network over the other. For example, assume that the interest $\Theta = \theta_1, \ldots, \theta_\tau$ over a set of τ topics corresponds to the functional essence of nodes. Then, in line with homophily [56], nodes are likely to create connections or to be in a close proximity if they share strong interest in a set $\vartheta \subseteq \Theta$ of topics.

Needless to say, it is necessary to account for the observed proximity. The third aspect is particularly important in cases where such proximity is only partially accessible or completely unaccessible. For example, the friendship and follower link structures of some users of social networks such as Facebook and Twitter is usually not accessible due to privacy constraints. In Wikipedia, the existing link structure between articles is incomplete due to the expensive manual effort involved to maintain it [62]. There is already a line of research that endeavors to improve the quality of the links by exploiting Wikipedia clickstreams or human navigation traces [62, 72, 84], considering them as possible signals for a hidden proximity between unlinked articles. For the case of social network users, processes like information diffusion events could play the role of the aforementioned signal.

In this study, attributes play the role of a functional essence of nodes and information diffusion events (cascades) that occur as a result of users interactions are used as indicators of an observed proximity. Intuitively, the main hypothesis behind the design of the proposed method in this chapter is as follows:

- if nodes have a strong attribute similarity, then they are likely to be in a close proximity;

- if nodes are in close proximity, then they are likely to have a strong interaction than random users farther away.

That is, we seek to develop an algorithm that preserves the topological proximity, attribute similarity and interaction history to learn a high-quality embeddings of nodes in an information network.

4.1 Summary of Contributions

In this study we propose a network representation learning algorithm based on topology, attributes and interaction histories. Earlier methods have focused on topology alone, and recent approaches revisited NRL by incorporating attributes. Although [82, 83] already noticed that diffusion events unveil interesting patterns about the topological orientation of nodes, existing studies in NRL have overlooked this insight. In this chapter, we examine how we can incorporate information diffusion events into NRL.

This is particularly useful when part of the network topology is missing. We took inspiration from previous studies who examine the correlation between the network topology and the dynamics of information diffusion events. That is, we take advantage of the aforementioned studies to indirectly account for missing links in network structure by exploiting diffusion events that have propagated over the entire network, including the non-accessible part. This gives us an edge over most existing studies, that require the presence of the full network structure.

A naive approach towards designing a NRL that uses topology, attributes, and information diffusion sources could be to learn independent representations from each of these sources, to later merge the learned representations. However, this fails to account for the correlation that may exist between the representations [55]. For this reason, we propose a novel method that jointly learn embeddings from the aforementioned three facets

of information networks.

We have carried out an extensive experimental evaluation of the proposed method using four real-world datasets, comparing it against state-of-the-art NRL techniques that are trained using the network structure only, across three kinds of network analysis problems.

4.2 Background and Problem

We use the graph definition proposed in Chapter 2, with uniform weights. That is, we consider a graph $G = (V, E, \mathbf{W}, a)$ where $(i, j) \in E \Rightarrow \mathbf{W}[\mathbf{i}, \mathbf{j}] = 1$. In addition, we also use cascades without timestamps, that is, a cascade $C = [(u_1, t_1), \ldots, (u_{|C|}, t_{|C|})]$ is transformed to $\mathit{strip}(C) = [u_1, \ldots, u_{|C|}]$. The set of cascades transformed by striping time information are denoted by $\mathcal{C}' = \{\mathit{strip}(C) : C \in \mathcal{C}\}$.

The last but not least information that we use are attributes; in this chapter, we consider textual attributes, that is, keywords associated to nodes. For a node $v \in V$, $a(v)$ corresponds to the set of keywords associated to v. Then we compute a new graph $G' = (V, E, \mathbf{W}')$ where $\mathbf{W}'$ is a new weight matrix and each entry $\mathbf{W}[\mathbf{i}, \mathbf{j}]'$ is defined as

$$\mathbf{W}[\mathbf{i}, \mathbf{j}]' = \frac{|a(i) \cap a(j)|}{|a(i) \cup a(j)|} \tag{4.1}$$

In other words, the weights correspond to the Jaccard similarity between the set of keywords of node i and j. Note that $\mathbf{W}'$ could be a very dense matrix if there is a popular attribute shared by all nodes. One can simply prune G' to reduce these kinds of trivial similarities, however in our datasets no such attribute exist. In a follow-up study [66] we have tackled this issue by modeling attributes as a separate bipartite graph.

We take into account interaction histories that are recorded as a reaction of users to others' post sorted according to their reaction time. This is

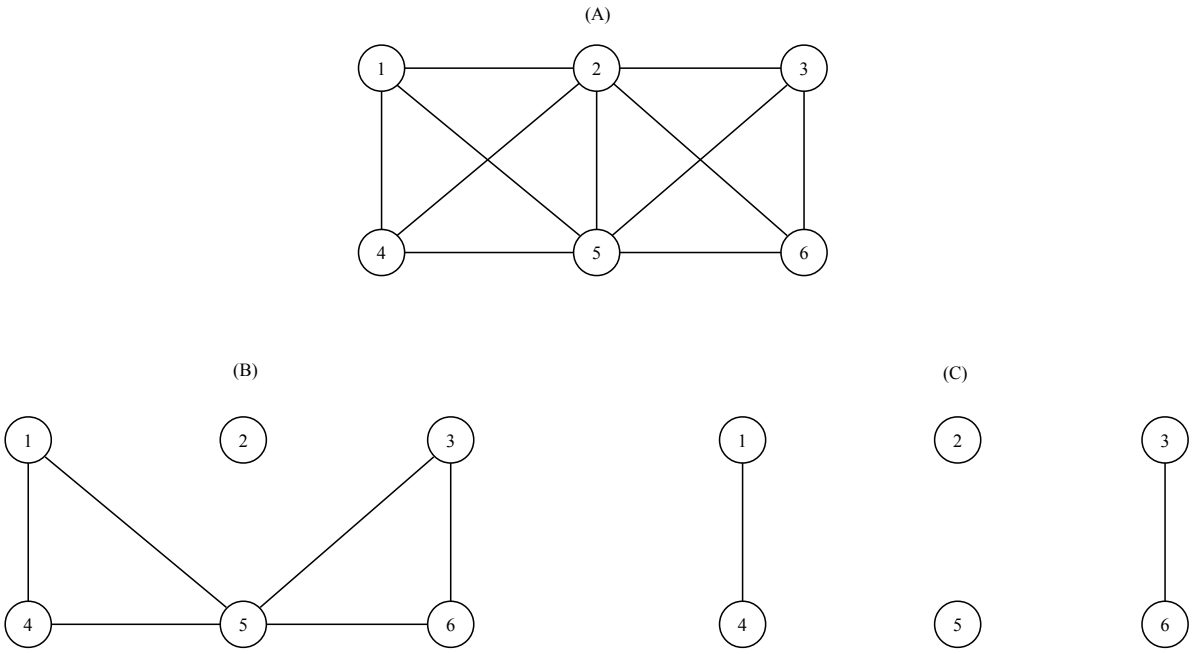


Figure 4.1: An example of a complete social network (A) and possible subgraphs (B) and (C) that could be crawled as a result of privacy settings of nodes. (B) If only node 2 have set its connection setting to private; and (C) if 3 have also decided to go private on its connections alongside 2.

exactly how we defined cascades in Chapter 2. Our assumption is that if a user generates a post, then it is more likely for another user in a close proximity to react to her post rather than a random user far away from the poster. In other words, we expect users in a close proximity to be involved in relatively similar cascades.

Suppose we have two disjoint communities, based on their interest in the Italian football club AC Milan and in Ethiopian politics. If a member of the AC Milan community posted a piece of content about the club, then its very likely for another member of this community to first react to the post than a member from the Ethiopian politics community. In general, we assume members of the Milan club community will have the tendency to appear in posts related to AC Milan and the Ethiopian community in posts related to the Ethiopian politics.

Furthermore, as stated earlier, cascades will also give us the advantage in case we only have a partially observed topology of the network. We claim that they play a vital role when only part of the true proximity is accessible and no other side information is available, for example due to privacy constraints in social networks.

If we examine the sample social network in Fig. 4.1, the true connections might look like as the one in (A). However, as a result of users privacy setting an interested third party in need of access to the social network could end up crawling different kinds of subgraphs. For instance, if only node 2 sets its connection to private, the most complete subgraph one can crawl is the one in (B); if both nodes 2 and 3 also decide to make their connections private, then one could only crawl at best the subgraph in (C). Clearly, the node embedding learned from subgraphs (B) and (C) could be rather far from the optimal one, and hence it is imperative to incorporate additional sources of information that provide a clue regarding the missing links.

In social networks like in Fig. 4.1, cascades can be extracted from hashtags, share and retweet activities, as shown in Table 4.1. We can exploit these to account for the missing links. Thus, even though one only has the subgraph in Fig 4.1 (B) or (C), a clever strategy could be devised to capitalize on possible patterns of interaction that could be associated with actual connections.

In this chapter, we only make a simple use of the cascades by combining them with sampled cascades. In Chapter 5 we shall demonstrate novel strategies to make a better use of the cascades for the NRL task. Now, we shall formally state the network representation learning problem that we seek to tackle in this chapter as follows:

Problem 1. Given a network $G' = (V, E, \mathbf{W}')$, a set of cascades $\mathcal{C}'$ with no time information, a dimension d, we seek to learn a representation of

Hashtag	**Users sorted according to infection timestamps**
$\#ht_1$	2,3,1,5,6
$\#ht_2$	3,4,2,1,5
$\#ht_3$	6,3,5,4,2
$\#ht_4$	1,6,5,3,4,2
$\#ht_5$	5,2,6,3
$\#ht_6$	1,3,4,5,2,6

Table 4.1: Cascades extracted from observed hashtag use of nodes of the social network in Fig. 4.1(A). A cascade is constructed by sorting nodes according to the time stamp that they have used a particular hashtag.

the network specified by $\Phi : V \rightarrow \mathbb{R}^d$, provided that Φ preserves nodes'

- topological information (observed proximity);
- functional essence (preferred attributes);
- interaction pattern (tendency to reacting to others' post).

Similar to [29], the aforementioned problem statement can be specified as solving the cost function

$$\mathcal{L} = loss(\text{SIM}(u, v; \mathbf{W}, a, \mathcal{C}'), \text{SIM}(u, v; \Phi)) \tag{4.2}$$

Recall that the attribute information is now captured by the weights on the edges, and hence Eq. 4.2 can be reformulated as

$$\mathcal{L} = loss(\text{SIM}(u, v; \mathbf{W}', \mathcal{C}'), \text{SIM}(u, v; \Phi)) \tag{4.3}$$

In this study, $\mathcal{L}$ is materialized using cross-entropy as in Eq. 4.4. That is, the deviation of the similarity – $\text{SIM}(u, v; \Phi)$ between any pair of nodes $u, v \in V$ in the output or representation space Φ from their similarity – $\text{SIM}(u, v; \mathcal{C}_{\mathcal{I}})$ in the input space $\mathcal{C}_{\mathcal{I}}$.

$$\mathcal{L} = -\sum_{u,v \in V} \text{SIM}(u, v; \mathcal{C}_{\mathcal{I}}) \log \text{SIM}(u, v; \Phi), \tag{4.4}$$

where the input space $\mathcal{C}_{\mathcal{I}}$ is a combination of the transformed cascades $\mathcal{C}'$ and a new set of simulated cascades $\mathcal{C}''$; more concretely,

$$\mathcal{C}_{\mathcal{I}} = \mathcal{C}' \cup \mathcal{C}''$$

We provide the details of the technique used to sample simulated cascades $\mathcal{C}''$ from $\mathbf{W}'$ in Section 4.3.1.

4.3 The Learning Algorithm

As the weighted network captures both topological and attribute information, from now on it should be understood when we refer to the structure in this chapter, attributes are implicitly referred to. Therefore, we propose an algorithm called MINERAL (**M**ulti**i**-modal **Ne**twork **R**epresent**a**tion **L**earning), inspired by an algorithm for language modeling called SKIP-GRAM.

As we have discussed in Chapter 3, the SKIPGRAM model is used to project words into a latent vector space. The projection is done by ensuring that the latent vectors of words that have the tendency to frequently appear in the same context are embedded close to each other.

In our problem setting, we want to achieve a similar goal and seek to project nodes to a latent vector space where nodes that:

- are in a close proximity
- have similar functional essence
- have the same interaction patterns

should be embedded close to each other. SKIPGRAM, however, is specialized for natural language documents. This introduces a challenge for our case, as the topological and functional essence of nodes are encoded in a

graph using G', which is not a sequence and can not be directly used in SkipGram. Therefore we propose a simulation of an information diffusion process to sample artificial cascades, which are sequence data.

4.3.1 Cascade Sampling

The goal is to sample cascades that capture nodes structural orientation in G'. Recall that the edges of G' encode attribute similarity. Intuitively, when sampling a cascade originating from a certain community, it should tend to stay within the community. There is a similar line of research [64, 28] that uses random walks for sampling sequences. In this study, we propose the simulation of information diffusion as it has been observed that the dynamics of diffusion processes reveal complex local and global structural patterns of the network. Therefore, we simulate the diffusion of influence or information using the independent cascade (IC) model [41].

Algorithm 1 shows the high-level steps required to generate cascades. We start by initializing $\mathcal{C}_{\mathcal{I}}$ with the observed cascades $\mathcal{C}'$ (line 1). Then, in line 2 for each node $i \in V$, r (line 3) cascades are sampled starting from i based on the IC model that is guided by the attribute similarity or edge weights $\mathbf{W}'$ of G' as an unnormalized probability of infection.

When simulateDiffusion(G', i, h) (line 4) is invoked, a cascade of size h is generated starting from node i. Let I_t denote the set of nodes infected at time step t; the simulation of a diffusion process works as follows:

1. At time step $t_1 = 0$, an artificial cascade sequence is initiated by infecting the current root, *i.e.*, $C'' = [i]$, and $I_1 = \{i\}$.

2. At time step t_l, for $l > 1$, each node $j \in I_{l-1}$ infected in the previous time step makes a single attempt to infect each of its outgoing neighbor $k \in out(j)$ [1] that is not already infected (i.e., $k \notin C''$), with

[1] $out(j)$ is used if the semantics of the edge (j, k) is influence flow; if not, infection attempts will be

Algorithm 1: CascadeGenerator(G', r, h)

```
1 C_I = C'
2 for i ∈ V do
3     repeat r times
4         C'' = simulateDiffusion (G', i, h)
5         C_I.insert(C'')
6 return C_I
```

a probability proportional to $\mathbf{W}[\mathbf{j}, \mathbf{k}]'$. If the infection succeeds, k is (i) appended to C'', *i.e.* $C'' = C'' \oplus [k]$, where $\oplus$ is a sequence concatenation operation, and (ii) included in I_l, *i.e.* $I_l = I_l \cup \{k\}$.

3. Proceed to the next time step t_{l+1} and repeat the process starting from step 2 while $|C''| \leq h$

After sampling each C'', it is added to $\mathcal{C}_\mathcal{I}$ (line 5) to build an input corpus. We restrict the size of cascades (the number of infected nodes) to be at most h nodes, because large, viral cascades (unlike non-viral ones) usually do not capture any relevant local or global structural relation of nodes [82, 83]. In other words, as the size of h gets larger, the probability of the diffusion process spreading throughout the network increases. This leads to cascade samples that have a mixture of nodes from multiple communities and that means introducing noise as cascade samples are less coherent in terms of their constituency.

Finally, once the sampled artificial cascades ($\mathcal{C}''$) are combined with the actual cascades($\mathcal{C}'$), we apply the SkipGram model on top of $\mathcal{C}_\mathcal{I}$ to learn the representation of nodes.

directed towards each incoming neighbor $k \in in(j)$

4.3.2 SkipGram formulation

The SkipGram model described in Section 3.3 is formulated by considering a natural language document corpus. In our setting, the document corpus corresponds to the set of combined cascades $\mathcal{C}_{\mathcal{I}}$ that we have just generated in the previous section. Therefore, instead of words, our focus is on nodes, and the goal is to embed them into a latent continuous vector space that preserves nodes structural proximity and interaction patterns. As we have already seen, this is exactly what the SkipGram model enables us to achieve.

To use this model, in the following we show how we can simply adopt the utility functions on cascades. The first is function $context(S, w_t, s)$; for cascades assuming a target node $C(t) = v_t$, it is re-defined as:

$$\begin{aligned} context(C, v_t, s) =& \{C(j) : 1 \leq t - s \leq j \leq t + s \leq |C|, j \neq t\} \\ & s.t.\ C \in \mathcal{C}_{\mathcal{I}} \end{aligned} \tag{4.5}$$

Therefore, the objective function of the SkipGram model based on the negative sampling for our problem is to minimize the log likelihood:

$$\min - \sum_{t=1}^{|C|} \sum_{v_c \in context(C, v_t, s)} \log P(v_c | \Phi(v_t)) \tag{4.6}$$

$$P(v_c | \Phi(v_t)) = \sigma(\Phi(v_c), \Phi(v_t)) + neg(v_t, \ell) \tag{4.7}$$

At this point, we are ready to train our model using the entire cascade corpus $\mathcal{C}_{\mathcal{I}}$ in exactly the same way as the SkipGram model in Section 3.3 of Chapter 3.

Dataset	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\|\mathcal{C}\|$	# labels	Type of labels
Twitter	595,460	14,273,311	397,681	5	top-5 communities
Memetracker	3,836,314	15,540,787	71,568	5	top-5 communities
Flickr	80,513	5,899,882	-	195	Groups
Blogcatalog	10,312	333,983	-	39	Interests

Table 4.2: Summary of the datasets

4.4 Experimental Evaluation

In order to demonstrate the effectiveness of our algorithm, we have carried out several experiments across multiple network analysis problems using multiple datasets.

4.4.1 Datasets

A brief summary of the datasets is given in Table 4.2.

- Twitter [82]: a dataset containing the follower network of Twitter users and cascade information of hashtags. Each time a user adopts a hashtag (by creating a new or using an existing one), it is added to the set of her keywords. A cascade is constructed by sorting the users according to their first use of a particular hashtag.

- Memetracker [51]: A dataset constructed by tracking news stories (memes) spread over mainstream medias and personal blog posts. A node corresponds to a website (mass media or personal blog) and cascades are constructed by tracking stories post across the websites over a period of nine months. A ground truth network structure is built based on hyperlinks found on the websites. If a given page A has a hyperlink to another page B, a direct link $A \rightarrow B$ is created to indicate that A has cited or referred B while discussing a certain story (meme).

- Blogcatalog [71]: a dataset containing a network of bloggers. There are 39 topic categories which are considered as content information for each author.

- Flickr [71]: a photo sharing site paired to a social network. Users place their pictures under a set of predefined categories which can be considered content information.

For Twitter and Memetracker, users are labeled based on their communities. First we identify the (non-overlapping) community to which a user belongs using [7], and then we associate it as her label. Only Twitter and Memetracker datasets have observed cascades. In addition, in all the experiments we have used $h = 500$ for Twitter and Memetracker, $h = 200$ for Flickr and $h = 50$ for Blogcatalog.

4.4.2 Baselines

Existing methods [85, 34] that consider content information are usually based on matrix factorization, which makes them not scalable for large networks. For this reason, we only consider the following two content-oblivious approaches as baseline methods:

1. DeepWalk [64]: is a method that utilizes truncated random walks for network embedding, where each step of a walk is chosen uniformly at random. Equivalent to the current work, they use the SkipGram model and it is trained using the walks.

2. Line [70]: is a proximity based approach, trained by concatenating two independently trained models based on the notions of first-order and second-order similarity of nodes. In other words, in the first phase they train a model that preserves the undirected link structure

Algorithm	P@100	P@500	P@1K	P@5K	P@10K	p@50K	p@100K	p@500K
MINERAL	**99.9**	99.8	99.8	99.8	**99.8**	**99.8**	**99.8**	**99.0**
DEEPWALK	96.6	97.0	97.1	97.1	97.1	97.1	97.1	96.9
Line	99.3	99.8	**99.9**	99.8	99.7	98.5	94.5	71.0

Table 4.3: Results for the link prediction task on the Twitter dataset

Algorithm	P@100	P@500	P@1K	P@5K	P@10K	p@50K	p@100K	p@500K
MINERAL	**100.0**	**99.9**	**99.9**	**99.6**	**99.5**	**99.5**	**99.4**	98.6
DEEPWALK	99.1	99.0	99.0	99.1	99.0	99.0	99.0	**99.0**
Line	91.2	92.2	89.9	85.2	83.3	72.8	68.9	65.4

Table 4.4: Results for the link prediction task on the Memetracker dataset

Algorithm	P@50	P@100	P@500	P@1K	p@5K	p@10K	p@50K	p@100K
MINERAL	**99.2**	**99.6**	**99.6**	**99.6**	**99.4**	**99.2**	**97.4**	94.9
DEEPWALK	96.6	96.6	97.4	97.5	97.5	97.5	**97.4**	**97.1**
LINE	54.4	61.0	61.6	58.8	51.6	48.9	44.2	42.5

Table 4.5: Results for the link prediction task on the Flickr dataset

between nodes; in the second phase, they train a model that preserves the directed or undirected 2-hop link structure of the network.

4.4.3 Link Prediction

One of the applications that we have introduced as an application of NRL algorithms is Link Prediction. Here we report the performance of algorithms for this task. As discussed in Section 1.3, the node embeddings learned by a NRL algorithm can be used instead of manually-crafted features. Towards this end, first we randomly sample 15% of the existing edges from the network; we also randomly sample the same amount of node pairs that are not in the edge set. We then use the learned embedding to effectively predict the links. That is, given a pair of nodes $\{u, v\} \subseteq V$, we compute the probability $p(u, v)$ of an edge existing between the two nodes

	Training Ratio								
Algorithm	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
MINERAL	**98.19**	**98.05**	**97.97**	**97.98**	**97.95**	**97.91**	**97.74**	**97.51**	**96.93**
DEEPWALK	97.78	97.76	97.86	97.67	97.61	97.45	97.42	97.02	96.01
LINE	84.19	85.74	85.02	85.11	85.18	84.69	84.06	82.20	76.19

Table 4.6: Node classification accuracy on different levels of labeled training set ratio for the Twitter dataset

	Training Ratio								
Algorithm	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
MINERAL	**98.19**	**98.05**	**97.97**	**97.98**	**97.95**	**97.91**	**97.74**	**97.51**	**96.93**
DEEPWALK	97.78	97.76	97.86	97.67	97.61	97.45	97.42	97.02	96.01
LINE	84.19	85.74	85.02	85.11	85.18	84.69	84.06	82.20	76.19

Table 4.7: Node classification accuracy on different levels of labeled training set ratio for the Memetracker dataset

as:

$$p(u, v) = \frac{1}{1 + e^{-(\Phi(u)^T \cdot \Phi(v))}}$$

Then we sort the predicted edges according to $p(u, v)$ in descending order and evaluate the performance of a NRL algorithm in correctly predicting the edges using the precision-at-K ($P@K$) score. $P@K$ measures the fraction of correctly predicted edges on the top-K results as

$$P@K = \frac{|\{u, v : (u, v) \in E \wedge rank(u, v) \leq K\}|}{K} \tag{4.8}$$

where $rank(u, v)$ is computed based on $p(u, v)$ score.

For each K value, we repeat the experiments 10 times and report the average. Tables 4.3, 4.4 and 4.5 show the results for the Twitter, Memetracker and Flickr datasets; MINERAL achieves a performance as good or better than the baselines in most of the experiments.

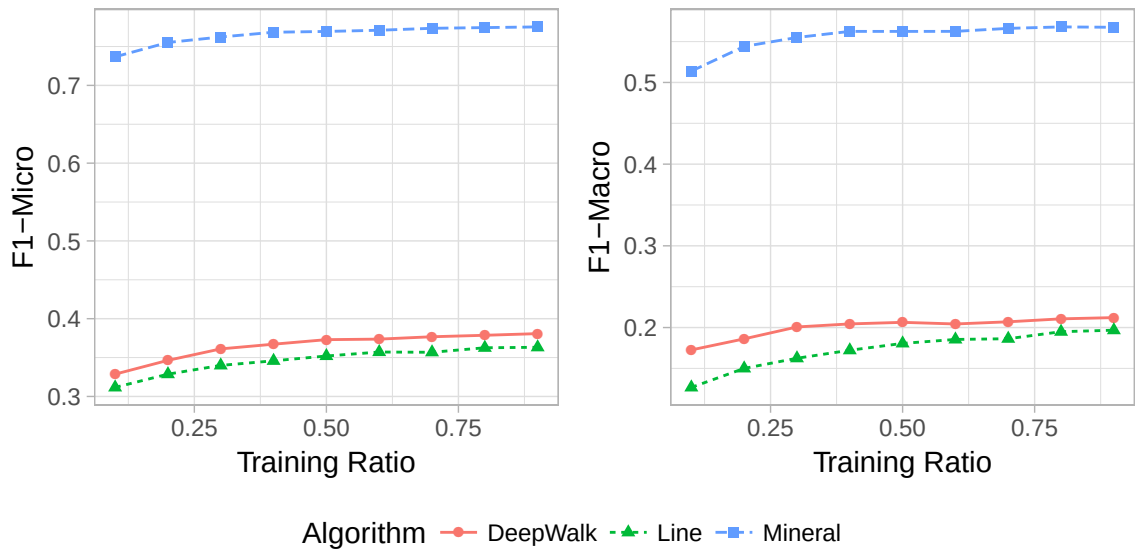


Figure 4.2: Multi-label classification (using one-vs-rest logistic regression classifier) on the Blogcatalog dataset

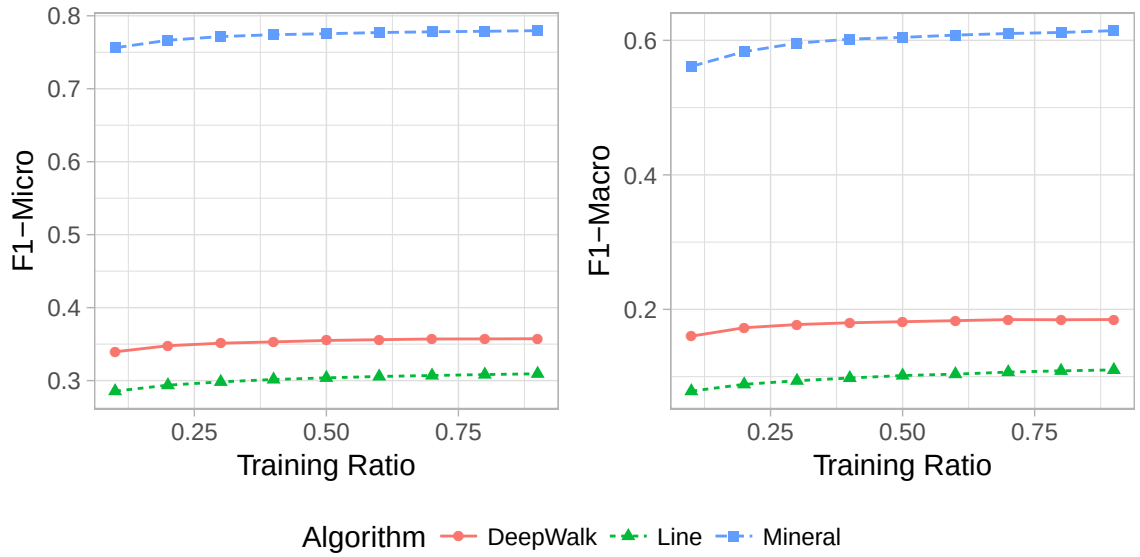


Figure 4.3: Multi-label classification (using one-vs-rest logistic regression classifier) on the Flickr dataset

4.4.4 Node Label Classification

The second application that we consider is node label classification. We consider two instance of it, namely multi-class and multi-label classifica-

tions. For the Twitter and Memetracker datasets, we tackled the multi-class classification problem, because – as shown in Table 4.2 – labels are communities and each node belongs to just a single community. In the other datasets, given that multiple labels are present, we performed multi-label classification. To evaluate the effectiveness of a model in the classification task, we adopt the same evaluation metrics as in previous studies, and hence we use Accuracy, F1-Micro and F1-Macro metrics.

The Multi-class classification results for the Twitter and Memetracker datasets are reported in Table 4.6 and 4.7, respectively. Similar to previous studies, we performed these experiments on different fractions of labeled training sets (Training Ratio $\in [10\% - 90\%]$, with a step of 10%). Under this setting, accuracy is the evaluation metric; and as shown in the tables, MINERAL performs slightly better than DEEPWALK and significantly better than LINE. For the other datasets, however, MINERAL significantly outperforms both baselines in multi-label classification. Figure 4.2 and 4.3 report the results on different training ratios (x-axis) using F1-Micro and F1-Macro measures (y-axis).

4.4.5 Network Visualization

The last but not the least application of NRL is network visualization. We use the Twitter dataset for this task, and the visualization is performed using t-Distributed Stochastic Neighbor Embedding (t-SNE) [74]. Given a set of q communities, an informative visualization should maintain a knit cluster for members of the same community and maintain clear boundaries between different communities. As shown in Fig. 4.4, MINERAL's visualization gives the best visual result. Members of each community are densely clustered and are far from members of other communities.

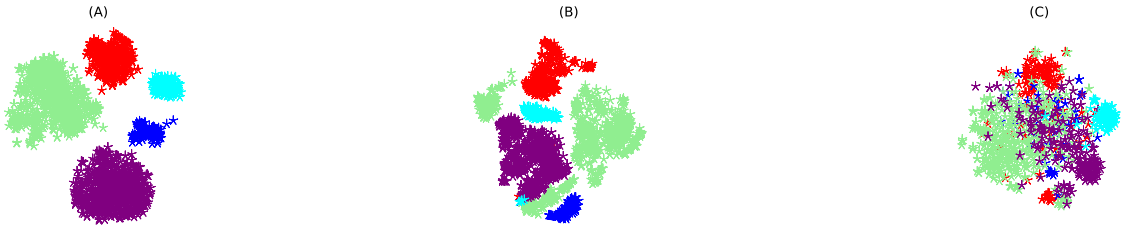


Figure 4.4: Visualization of top-5 communities with at most 2000 users in the Twitter Dataset using (A) MINERAL (B) DEEPWALK and (C) Line

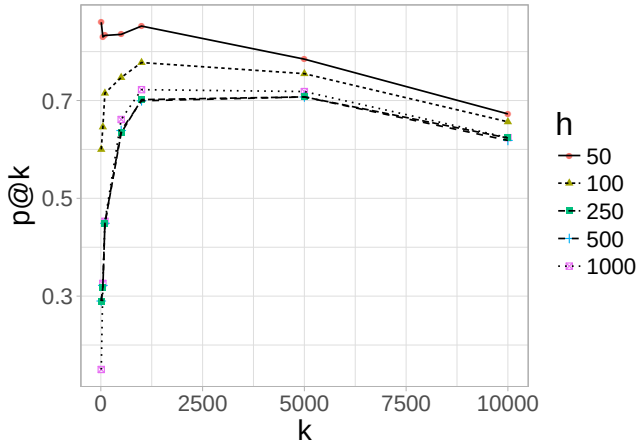


Figure 4.5: Sensitivity of the parameter h using the link prediction task on Blogcatalog

4.4.6 Parameter Sensitivity

To conclude the chapter, we analyze the sensitivity of the hyper-parameters of the model, namely r and h, controlling the number and length of cascades to sample, respectively. Recall that earlier we have argued that its sufficient to sample truncated cascades. As shown in Fig 4.5, the precision-at-K significantly drops as we increase the size of h.

For example, for a fixed $K = 10$, the precision-at-K is $P@K = 0.86$ for $h = 50$, $P@K = 0.6$ for $h = 100$, $P@K = 0.29$ for $h = 500$, and $P@K = 0.15$ for $h = 1000$. This is caused by the introduction of noise as a result of increasing h. Because as we increase h to very large values the likelihood of sampling unrelated neighbors also increase.

Chapter 5

Network Representation Learning without Structural Information

Most studies on network representation learning are based on the assumption that the structure of the network is known. However, as we have already argued in the introduction and in the previous chapter, this is not always the case and there are several instances where one might lack partial or complete information about the structure [27, 26, 18].

Not all hope is lost, though: even though the underlying network over which a physical or virtual process takes place may be unknown, we can often record or reconstruct traces of events that occurred over the network [26, 52]. For example, it is relatively simple to acquire traces of public share events from OSNs, or keep record of infection events during an epidemic. We can use such kinds of events as a window to look inside the actual network.

The considerable research work [27, 26, 18, 38, 46] towards reconstructing or inferring the hidden network from the diffusion events that have occurred over this network could be taken as motivation to what we seek to achieve in this chapter.

In fact, very recently several papers have shown that such inference is indeed possible using node embeddings that are learned directly from

diffusion events [46, 38] or from partially observed network structure [87].

Therefore, it is particularly important to design algorithms to effectively and efficiently learn representations of nodes when the underlying network structure is completely unknown, by exploiting information about cascades. This chapter introduces the NetTensor framework, which responds to this need.

With respect to existing works that propose to learn a network representation based on completely or partially missing information about the network structure [46, 38, 87], NetTensor provides a full-fledged solution for NRL, capable to exploit multiple types of information extracted from cascades and applicable to several different problems, such as link prediction, network reconstruction and node classification. Because of this, we compare it against three strong baselines in the field of NRL, namely DeepWalk [64], Node2Vec [28] and Line [70]. Clearly, the proposed comparison is unfair, as all the baselines are based on the full knowledge of the structure of the network. Our goal is thus not to show that NetTensor outperforms the baselines (an inherently impossible goal for NetTensor), but rather show that it does an "excellent" job in solving the aforementioned problems.

Properties and roles of nodes are well-defined when the network topology is known. Contrarily, when the topology is hidden, the first challenge is to model nodes in order to approximate their true neighborhood or proximity information and any other relevant features. Towards this end, we revisit previous findings on the correlation between properties of the network structure and interaction patterns in cascades that occur over such networks. Then, we propose novel techniques to model nodes, so as to estimate nodes proximity and extract relevant node features by exploiting cascades. Therefore, in Sections 5.2- 5.3 we introduce the node proximity models and feature extraction techniques.

5.1 Summary of Contributions

When the topology of a network is known, one can easily extract proximity of local neighborhood information regardless of the computational cost. However, when the topology is hidden, the aforementioned task is not straightforward. Therefore, in this chapter we propose an array of techniques that can be used to extract nodes proximity and features that are indirectly related to the network topology.

We have experimentally evaluated and shown that these techniques perform well in different kinds of network analysis problems by comparing them against the state-of-the-art NRL techniques that require topology information.

5.2 Node Proximity Models

For modeling nodes proximity, we propose two paradigms, one *delay-aware* and another *delay-agnostic*. In the former, nodes are modeled according to their reaction time with respect to the other nodes in cascades. In the latter, we simply model nodes according to their order of appearance in cascades, irrespectful of time.

As we have been arguing in the previous chapter, we also assume that if nodes have shared functional essence, *i.e.*, share attribute information or interest in related topics, they are likely to interact with each other. Based on this assumption, we propose three kinds of feature extraction techniques, yet again using cascades only.

The node modeling techniques are inspired by observations from existing studies that have empirically shown that nodes that are closely interconnected (e.g., directly connected or sharing community membership) are likely to cause *complex contagions*, i.e. non-viral events that lead to

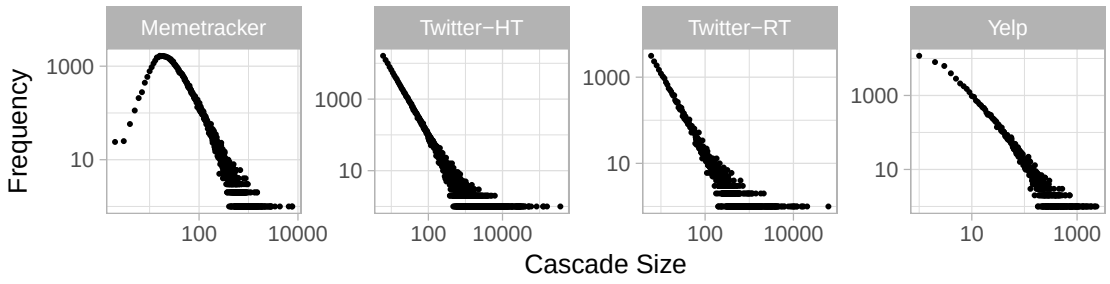


Figure 5.1: Cascade size distribution for the datasets used in our experiments

cascades that are trapped within a community [82, 83]. Interestingly, diffusion events in social networks are known, often, for following a long-tail distribution (powerlaw, log-normal) as shown in Fig. 5.1; i.e., most of them spread like complex contagions.

Besides, the node modeling techniques are inspired by the assumption that nodes are likely to appear close to each other (in terms of time and position) in diffusion events if they share attributes or interest in related topics or in general if they can influence each other (that is, if they are in a "close" proximity in the underlying social network). This assumption is similar to those proposed by [26, 27, 18, 86]. In other words, close proximity (homophily) in the social network is likely to lead to frequent and close co-occurrences in cascades.

These are all desirable properties, as they allude to a direct relation between the properties of nodes in the network and their interaction patterns in diffusion events, as shown by the *causality* relations in Fig. 5.2. Thus, if the learning goal of a NRL method is to preserve proximity, attribute similarity, interest in topics (as shown in the blue arrow), equivalently one has to preserve different interaction patterns such as response times, closeness and frequency of interactions of nodes in cascades (as shown in the red arrow) while learning nodes representation without knowing their actual topology.

In the following delay-aware and delay-agnostic node proximity models

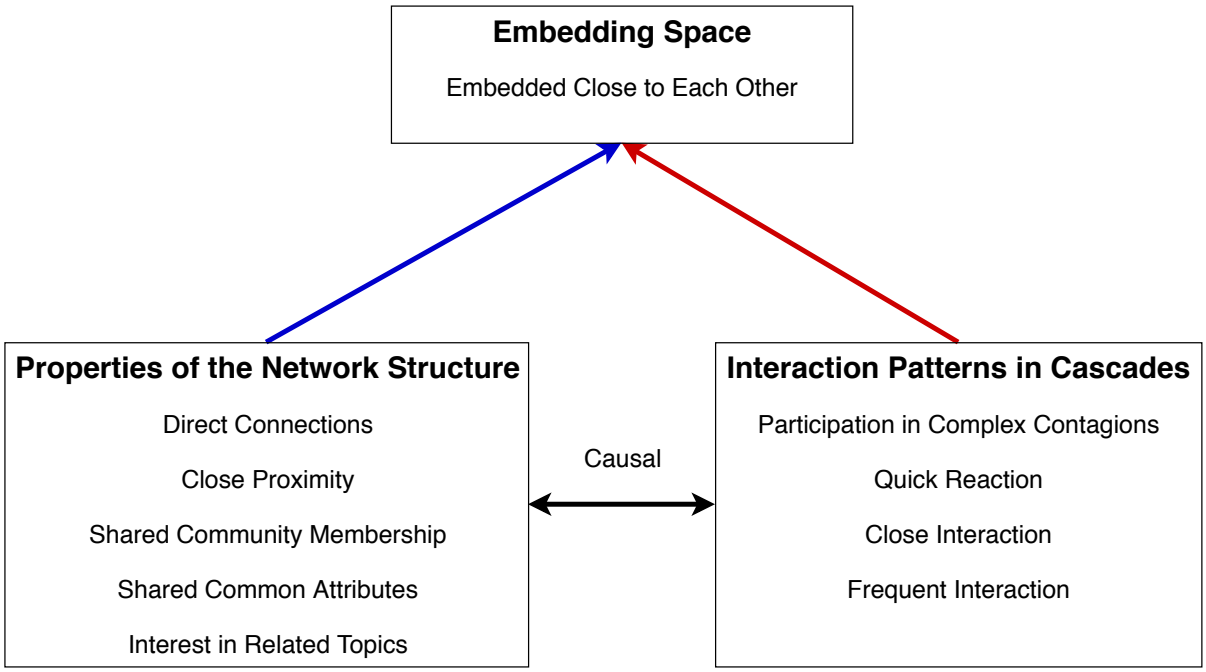


Figure 5.2: Relations between properties of a network structure, interaction patterns in diffusion events and an embedding space

we intend to estimate nodes proximity from cascades.

5.2.1 Delay-Aware Node Proximity Models

When a node posts a piece of content, such post first has to reach the immediate neighbors before reaching the farthest part of the network. For example, consider Fig. 5.3: if node i has created a post, node a can not see/share the post unless node e shares it first, and similarly node v has to wait until either node j or u shares it before she can share/see it. In addition, before the post is visible to the members of another community – e.g. C_2, first, it has to get to node v that bridges community C_2 with C_1 (where the post has originated from).

The main assumption behind our model of node proximity follows this observation: if a pair of nodes are closely connected in the original network, either directly or by belonging to a community, then it is more likely

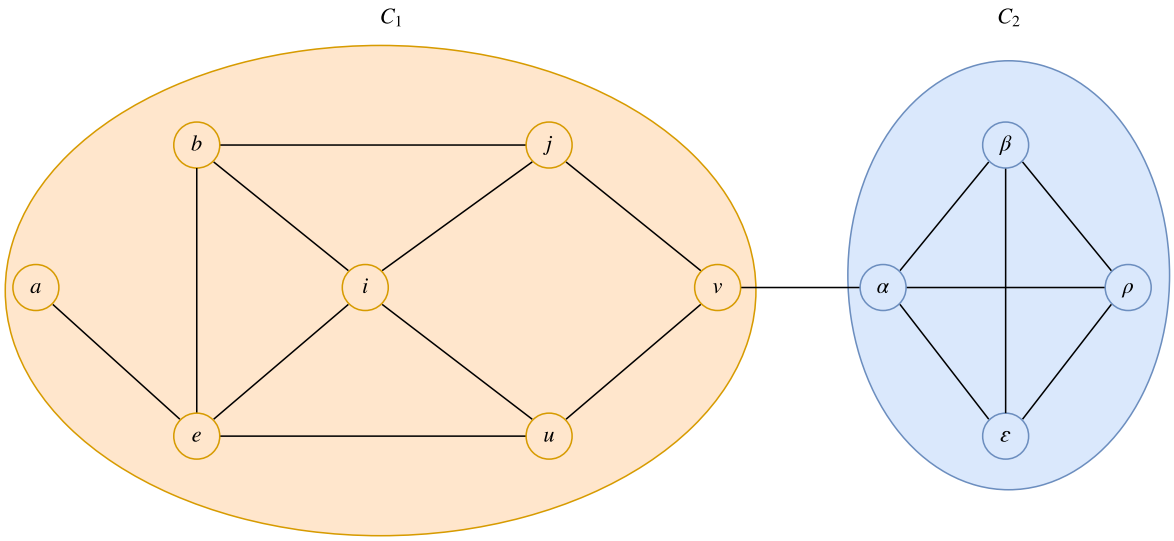


Figure 5.3: An example graph with two communities, C_1 and C_2.

for these nodes to quickly react to posts from either of them rather than reacting to posts coming from a random node farther away.

Therefore, if we observe a tendency among pairs of nodes to appear close to each other in cascades, we consider that as a strong signal for a close proximity in the hidden network.

We define a delay-aware proximity vector that associates each node u with an n-dimensional *pairwise proximity vector* $\mathbf{p_u}$, where $\mathbf{p_u}[\mathbf{v}]$ is a non-symmetric proximity value between node u and any other node $v \in V$.

To compute $\mathbf{p_u}$, we first compute a reaction time summary over all the cascades contained in $\mathcal{C}$, as follows. Let $C \in \mathcal{C}$ be a cascade, and let i, j be two indexes such that $C(i) = (u_i, t_i)$ and $C(j) = (u_j, t_j)$, with $1 \leq i < j \leq |C|$ and thus $t_i < t_j$. The *reaction time* between the events associated to u_i and u_j in C is the distance between the two timestamps:

$$dist_C(u_i, u_j) = \begin{cases} t_j - t_i & t_j \geq t_i \\ +\infty & t_j < t_i \end{cases} \tag{5.1}$$

Next, we compute the pairwise proximity $prx_C(u_i, u_j) \in [0, 1]$ with re-

spect to C as:

$$prx_C(u_i, u_j) = \frac{1}{1 + dist_C(u_i, u_j)} \tag{5.2}$$

The higher $prx_C(u_i, u_j)$ is, the smaller is the reaction time between t_i and t_j. Working with $prx_C(u_i, u_j)$ is mathematically more convenient than working with $dist_C(u_i, u_j)$; it leads to a very sparse matrix whose sparsity can be leveraged to reduce the memory footprint and the computational cost. Note that $prx_C(u_i, u_j) \neq prx_C(u_j, u_i)$; if u_j react after u_i, by definition u_i does not react after u_j and thus $dist_C(u_j, u_i) = \infty$ and $prx_C(u_j, u_i) = 0$.

We obtain $\mathbf{p_u}[\mathbf{v}]$ by aggregating the proximities over all the cascades containing both u and v:

$$\mathbf{p_u}[\mathbf{v}] = \sum_{C \in \mathcal{C}_u \cap \mathcal{C}_v} prx_C(u, v) \tag{5.3}$$

The complete proximity matrix $\mathbf{P} = [\mathbf{p_1}, \ldots, \mathbf{p_n}]^T$, computed over all the cascades, is obtained by combining the individual proximity vectors. The row vector $\mathbf{P}[\mathbf{i}]$ and the column vector $\mathbf{P}[\mathbf{i}]^T$ model node i's proximity from two different perspectives, i.e. after and before i has been infected, respectively. Consequently, it is worth to note that the matrix $\mathbf{P}$ is not symmetric, since $\mathbf{P}[\mathbf{i}] \neq \mathbf{P}[\mathbf{i}]^T$.

This property has proved to be useful for predicting influence propagation probabilities in the independent cascade (IC) model [8, 41]. As shown later, we capitalize on such property to propose a unified model to learn influence propagation probabilities for the IC model. The computational complexity of computing $\mathbf{P}$ is quadratic in the length of cascades and linear in the number of cascades: $O(c \cdot \ell^2)$, where $\ell = \max\{|C| : C \in \mathcal{C}\}$.

5.2.2 Delay-Agnostic Node Proximity Model

The delay-agnostic node proximity model is similar to the delay-aware variant and is established under the same assumptions. It differs because

the timestamp are discarded and only the time-induced order of nodes in each cascade is used. The distance computation in Eq. 5.1 is replaced by

$$dist_C(u_i, u_j) = \begin{cases} j - i & j \geq i \\ +\infty & j < i \end{cases} \tag{5.4}$$

and hence instead of reaction time, *dist* captures the order of nodes.

Similarly as before, the distances are used as building blocks of the proximity matrix $\mathbf{P}$; this time, however, such matrix is delay-agnostic. Such modeling will complement the aforementioned approach by utilizing the order of infection, particularly when there is a lack of pattern in the reaction time of nodes.

5.2.3 Window-Based Pairwise Proximity Model Optimization

Following our assumption on the tendency of similar nodes to quickly react to each others post and the empirical observation that most cascades occur between similar nodes [82, 83], we propose a window-based optimization for the pairwise node modeling strategy, for both the delay-aware and the delay-agnostic approaches.

Given a model parameter pw that specifies the size of proximity window, we only consider the pairwise proximity measure between nodes within a sliding window, instead of considering the entire collections of nodes involved in the cascade. Let

$$C = [(s, 0), (t, 2), (u, 3), (v, 4), (w, 9), (x, 20), (y, 30), (z, 50)]$$

be a cascade and $pw = 4$ a window size. In the original pairwise proximity models, we would need to compute the proximity for each pair of nodes in

$$C' = (s, t, u, v, w, x, y, z)$$

that is, $8^2 = 64$ operations.

In the window-based version, we have to evaluate the pairs in each of the subsequences of the set $\mathcal{S}$

$$\mathcal{S} = \{S_1 = (s,t,u,v), S_2 = (t,u,v,w), S_3 = (u,v,w,x),$$
$$S_4 = (v,w,x,y), S_5 = (w,x,y,z)\}$$

that are generated by a sliding window of $pw = 4$; and the computational cost is proportional to $O(pw^2 * \ell)$, that is, $4^2 \cdot 8 = 128$ operations. Obviously such cost is incurred due to redundant computations that can be avoided. For example the set of pairs $\{(t,u),(t,v),(u,v)\}$ in the evaluation of S_2 and $\{(u,v),(u,w),(v,w)\}$ in the evaluation of S_3 would have been already processed during the evaluation of S_1 and S_2 respectively. Similarly, the evaluation of S_3, S_4 and S_5 contains redundant computations.

In general, computing the i^{th} sequence S_i will require the extra cost of evaluating

$$\binom{pw-1}{2} - pw + 1$$

number of pairs that have already been computed during the evaluation of S_{i-1}. In total,

$$(|C'| - pw + 1) \cdot (\binom{pw-1}{2} - pw + 1)$$

repeated operations are required to evaluate C; for the entire set of cascades $\mathcal{C}$, there is an extra asymptotic cost

$$c \cdot (\ell - pw + 1) \cdot (\binom{pw-1}{2} - pw + 1)$$

Upon a simple probing of each sequence, we realize that in the i^{th} sequence S_i for $i > 1$ only the last node, *i.e.* $y = S_i[|S_i| - 1]$, is the new one; the rest are dragged from S_{i-1}, and that is what caused the repeated evaluations. Such repetitions can be avoided with a simple trick. We only perform a pairwise computation for the first subsequence S_1, which requires

pw^2 operations. For the remaining subsequences $\mathcal{S}_{>1} = \{S_i \in \mathcal{S} : i > 1\}$ we only have to generate $pw-1$ pairs $\{(x, y)_k : k = 1, \ldots, pw-1\}$ between the last element y and the remaining elements $[S_i[j] = x : 0 \leq j < |S_i| - 1]$ of S_i and this cost is proportional to $(\ell - pw + 1) \cdot pw$. The overall asymptotic cost will then be reduced to $O(c \cdot (pw^2 + \ell \cdot pw))$, that is, $4^2 + 8 \cdot 4 = 48$ operations for the above example.

5.3 Node Feature Extraction

The pairwise proximity models proposed above are intended to capture the global proximity of all nodes. Here, we seek to extract additional features that capture both global and local properties of a node. Towards this end, we propose three kinds of feature-extraction techniques from cascades, based on co-occurrence, local neighborhood and topic features.

5.3.1 Statistical Feature Extraction

Computing statistical features from a document corpus is one of the most widely used strategies to acquire syntactic and semantic relation between words. In the case of graphs, the co-occurrence feature extraction technique computes co-occurrence statistics between nodes. It is based on the simple assumption that if two nodes tend to frequently co-occur in cascades, there is some latent similarity between them. This is a global feature as we have to compute pairwise co-occurrence counts. We use $\mathbf{cf_u} \in \mathbb{R}_+^n$ to denote the *co-occurrence feature* vector of u where $\mathbf{cf_u}[\mathbf{v}] = |\mathcal{C}_u \cap \mathcal{C}_v|$ is the co-occurrence between node u and v. Note that $\mathbf{cf_u}[\mathbf{v}] = \mathbf{cf_v}[\mathbf{u}]$, and hence $\mathbf{CF} = [\mathbf{cf_1}, \ldots, \mathbf{cf_n}]^T$ is a symmetric matrix. The computational cost of computing $\mathbf{CF}$ is similar to that of $\mathbf{P}$; the advantage, however, is that we can compute both $\mathbf{P}$ and $\mathbf{CF}$ simultaneously using the window-based optimization in Section 5.2.3.

5.3.2 Local Feature Extraction

Inspired by [29] who proposed to aggregate neighborhood features of a node (such as the degree of its neighbors) in order to somehow capture its local context, we propose here to capture the local context of nodes directly from the cascades. As before, the assumption is that a node and its local neighbors (direct connections) are more likely to appear close to each other in cascades.

Towards this end, we utilize a method known as SkipGram [57] taken from the NLP area. As we have discussed in Chapter 3, SkipGram projects words in a document corpus into a vector space in such a way that their distributional semantics is preserved. Equivalently, our goal here is to use the context information of nodes in cascades just like words in documents. This is similar to the technique that we have used for Mineral in Section 4.3. Note that, unlike the proximity models, the features computed in this manner are oblivious to the order of nodes appearance in cascades. We use an l–dimensional vector $\mathbf{lf_u} \in \mathbb{R}^l$ to denote the context feature of node u, and $\mathbf{LF} = [\mathbf{lf_1}, \ldots, \mathbf{lf_n}]^T$.

One might face a situation where the quality of the extracted features is poor due to an insufficient number of cascade samples. Nonetheless, as reaction-time and order are not relevant for this technique, we propose an optional order- and time-oblivious sequence sampling phase as follows.

First, we build a user-cascade bipartite graph from users to the cascades as shown in Fig. 5.4, connecting every user to every cascade that the user is involved in. Then, we sample a sufficient number of sequences from the bipartite graph by simulating multiple truncated random walks over the user-cascade graph [64]. Note that this network should not be confused with the actual interaction network, as we are merely utilizing just the cascades to build it. Once a sufficient number of sequences are sampled,

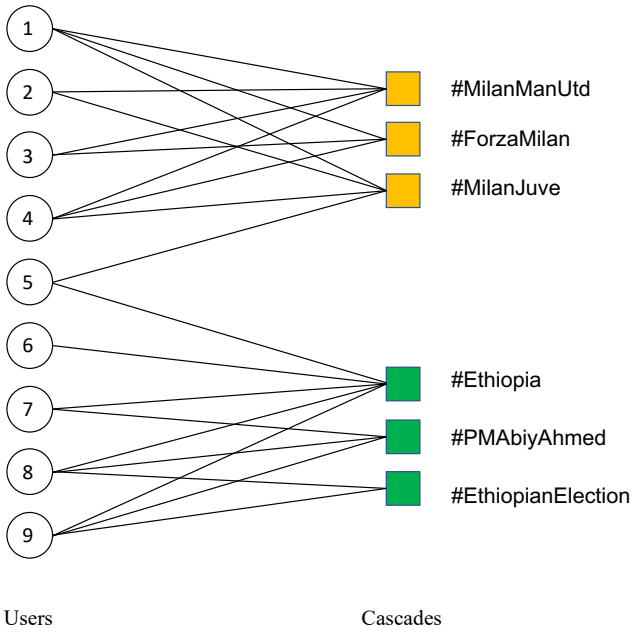


Figure 5.4: User-cascade bipartite graph illustration. Two groups of users discussing about the AC Milan football club and Ethiopian politics

we apply the SkipGram model on the sampled sequences instead of the cascades and extract the relevant features.

5.3.3 Topic Feature Extraction

One of the most widely used feature in classical NLP is the term-document matrix that is constructed from TF-IDF (term frequency-inverse document frequency) weights. The matrix is constructed by computing the frequency (TF) of each word within a given document and the inverse of the word's frequency across documents (IDF) in a given corpus. Several topic detection algorithms, such as the Latent Semantic Indexing (LSI) [14], Latent Dirichlet Allocation (LDA) [6] and Non-Negative Matrix Factorization (NMF), are normally executed over such matrix.

Inspired by this, we consider cascades as documents and nodes as words and we build a node-to-cascade matrix. To build this matrix, however, we

utilize infection events instead of TF-IDF. As in the case of the proximity models, we employ both delay-aware and delay-agnostic events. Thus for each node u we build a c–dimensional event vector $\mathbf{e_u} = [\mathbf{e_u}[\mathbf{1}], \ldots, \mathbf{e_u}[\mathbf{c}]]$, based on the time or order of node u's infection events in all the cascades. Each entry $\mathbf{e_u}[\mathbf{i}]$, for $1 \leq i \leq c$, is associated with the infection time or order of u in the i^{th} cascade. Given the i^{th} cascade C and $j : 1 \leq j \leq |C|$, let $C(j) = (u_j = u, t_j)$; then $\mathbf{e_u}[\mathbf{i}] = t_j$ and $\mathbf{e_u}[\mathbf{i}] = j$ are the components of the delay-aware and delay-agnostic vectors, respectively. If a node never occurs in the i^{th} cascade, then $\mathbf{e_u}[\mathbf{i}] = \infty$, and the event matrix is $\mathbf{E} = [\mathbf{e_1}, \ldots, \mathbf{e_n}]^T$. The computational complexity of constructing $\mathbf{E}$ is $O(c \cdot \ell)$, as we only need to scan each cascade once.

Here, our intuition is that if two users are highly interested in some topic-like latent structure θ, then they are more likely to have a close (in terms of time or order) interaction pattern than another random user who is less interested in the topic. We assume that the cascades were generated as a result of discussion over a set of τ topic-like latent structures $\vartheta = \{\theta_1, \ldots, \theta_\tau\}$.

Therefore, given the number of topics τ and the event matrix $\mathbf{E}$, we compute a topic-like feature matrix $\mathbf{TF} = \Theta(\mathbf{E}; \tau) \in \mathbb{R}^{n \times \tau}$, where Θ can be any topic modeling algorithm such as LDA and NMF. In this work we have experimented with Neural-Non-Negative Matrix Factorization (NNMF) and *Truncated Singular Value Decomposition* (TSVD), which we have discussed in Chapter 3. We have observed a small qualitative gain by using NNMF, however TSVD is more computationally efficient (by more than an order of magnitude).

Relation to Local Context Features This approach is closely related to the local feature extraction technique in the previous section; it assumes, however, that similarity in the latent structure is dependent on the order

and/or reaction times. To shed more light on these models and their relation, let us consider the following illustration under the delay-agnostic paradigm.

Suppose we are given the following set of cascades $\mathcal{C}$

$$\begin{aligned}\mathcal{C} = \{&C_1 = [(u_1 = 1, t_1), (u_2 = 4, t_2), (u_3 = 3, t_3), (u_4 = 2, t_4)],\\ &C_2 = [(u_1 = 1, t_1), (u_2 = 4, t_2), (u_3 = 5, t_3)],\\ &C_3 = [(u_1 = 1, t_1), (u_2 = 4, t_2)],\\ &C_4 = [(u_1 = 1, t_1), (u_2 = 4, t_2), (u_3 = 5, t_3), (u_4 = 2, t_4), (u_5 = 3, t_5)],\\ &C_5 = [(u_1 = 8, t_1), (u_2 = 7, t_2), (u_3 = 5, t_3), (u_4 = 6, t_4), (u_5 = 9, t_5)],\\ &C_6 = [(u_1 = 8, t_1), (u_2 = 7, t_2)],\\ &C_7 = [(u_1 = 8, t_1), (u_2 = 7, t_2), (u_3 = 9, t_3), (u_4 = 6, t_4)]\\ &C_8 = [(u_1 = 8, t_1), (u_2 = 7, t_2), (u_3 = 5, t_3), (u_4 = 9, t_4), (u_5 = 6, t_5)]\}\end{aligned}$$

In the delay-agnostic paradigm, only the time induced order of infection events is relevant, therefore for ease of readability we can simplify the set of cascades $\mathcal{C}$ as

$$\begin{aligned}\mathcal{C} = \{&C_1 = [1, 4, 3, 2], C_2 = [1, 4, 5], C_3 = [1, 4], C_4 = [1, 4, 5, 2, 3]\\ &C_5 = [8, 7, 5, 6, 9], C_6 = [8, 7], C_7 = [8, 7, 9, 6], [8, 7, 5, 9, 6]\}\end{aligned}$$

As discussed in Section 5.3.2, we can then easily construct the bipartite graph in Fig. 5.5 (A) and use it to extract the *local context* features. We can also formulate the bipartite graph as a binary *user-cascade* weight matrix $\mathbf{W}$ as in Eq. 5.5.

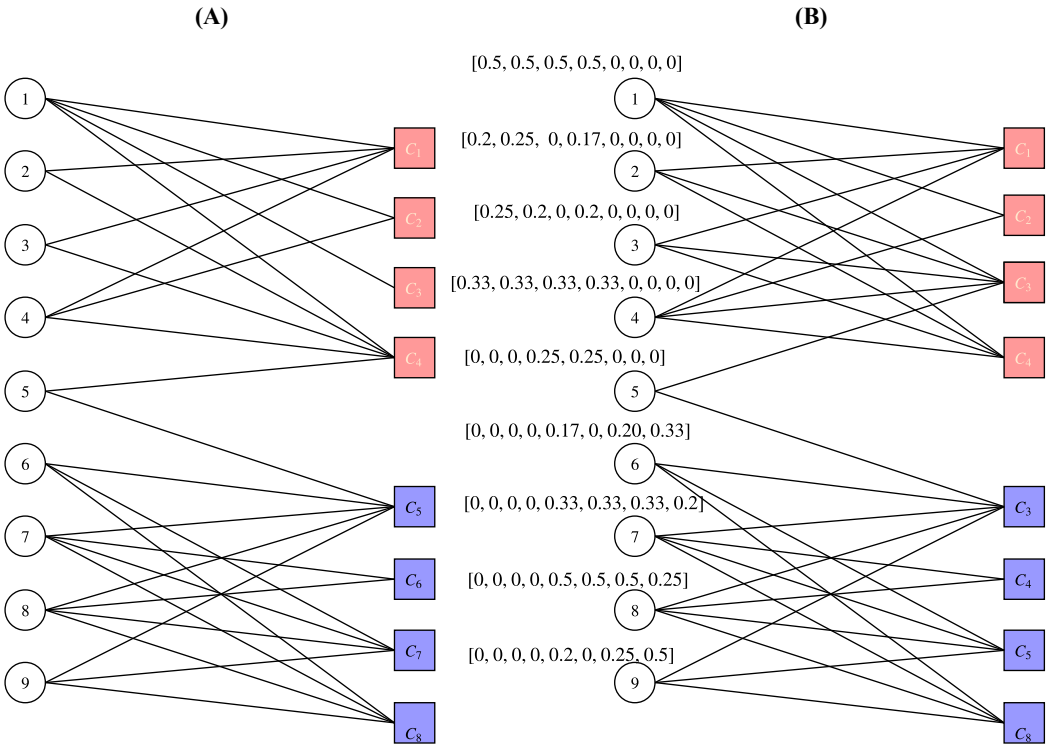


Figure 5.5: Relations between *local context* and *topic* feature extractions. The former method uses a bipartite graph in (A). Similarly the latter one can be modeled as a weighted bipartite graph by taking the rows of the transformed event matrix $\mathbf{E}'$ to put weights on the edges

$$\mathbf{W} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{5.5}$$

Each row $\mathbf{W}[\mathbf{u}]$ of $\mathbf{W}$ encodes user u's participation in each cascade $C_i \in \mathcal{C}$

according to the following rule

$$\mathbf{W}[\mathbf{u}, \mathbf{i-1}] = \begin{cases} 1, & \text{if } \exists j : 1 \leq j \leq |C_i|, C_i(j) = (u_j = u, t_j) \\ 0, & \text{otherwise} \end{cases}$$

Earlier in this section, we have seen the method used to construct the event matrix $\mathbf{E}$ in Eq. 5.6. Nonetheless, working directly on $\mathbf{E}$ is not mathematically convenient; Section 5.4 shows how it can be transformed for practical purpose. For the moment, suppose we have applied the desired transformation and obtained the matrix $\mathbf{E}'$ in Eq. 5.7, which has an equivalent semantics as $\mathbf{E}$ but in an opposite way. Then, it is straightforward how one can easily build a weighted user-cascade bipartite graph as in Fig. 5.5(B) based on $\mathbf{E}'$.

$$\mathbf{E} = \begin{pmatrix} 1 & 1 & 1 & 1 & \infty & \infty & \infty & \infty \\ 4 & 3 & \infty & 5 & \infty & \infty & \infty & \infty \\ 3 & 4 & \infty & 4 & \infty & \infty & \infty & \infty \\ 2 & 2 & 2 & 2 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 3 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 4 & \infty & 4 & 2 \\ \infty & \infty & \infty & \infty & 2 & 2 & 2 & 4 \\ \infty & \infty & \infty & \infty & 1 & 1 & 1 & 3 \\ \infty & \infty & \infty & \infty & 5 & \infty & 3 & 1 \end{pmatrix} \tag{5.6}$$

$$\mathbf{E}' = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0.2 & 0.25 & 0 & 0.17 & 0 & 0 & 0 & 0 \\ 0.25 & 0.2 & 0 & 0.2 & 0 & 0 & 0 & 0 \\ 0.33 & 0.33 & 0.33 & 0.33 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.17 & 0 & 0.20 & 0.33 \\ 0 & 0 & 0 & 0 & 0.33 & 0.33 & 0.33 & 0.2 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0.25 \\ 0 & 0 & 0 & 0 & 0.2 & 0 & 0.25 & 0.5 \end{pmatrix} \tag{5.7}$$

Finally, if we simply apply any kind of matrix factorization or representation learning algorithm over $\mathbf{W}$ and $\mathbf{E}'$, we get the two embeddings in Fig. 5.6.

Pertinent to their participation in cascades, exactly as it is shown in

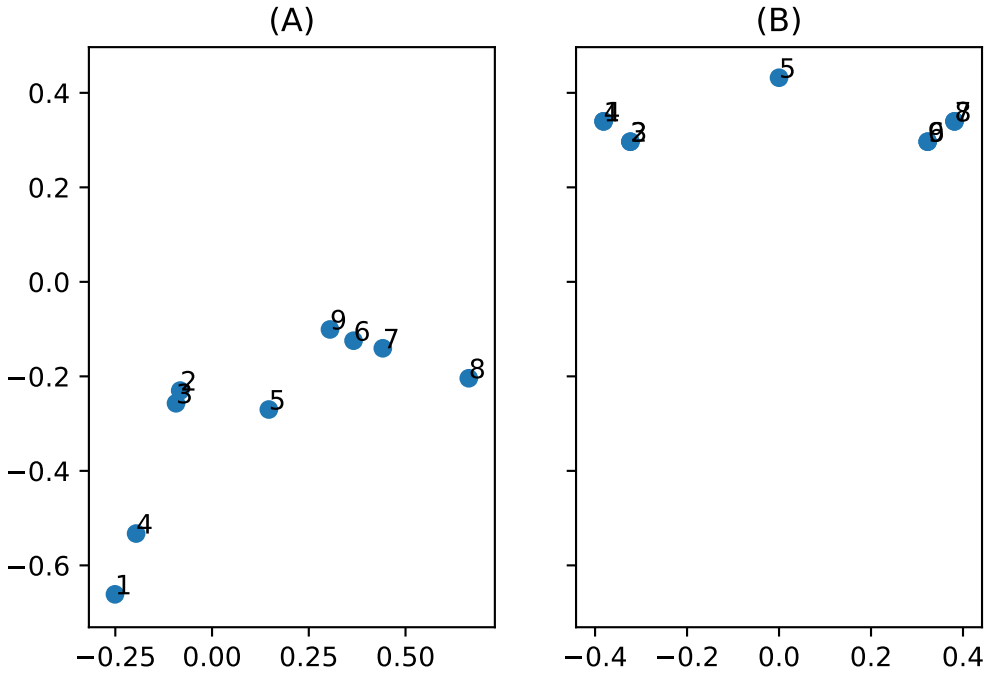


Figure 5.6: Topic (A), where order matters, vs. local context (B) features, where order does not matter. (A) is plotted form **LF** and (B) from **TF**.

Fig. 5.5(A), the node embeddings computed from $\mathbf{W}$ are clustered into two sets of cohesive groups as shown in Fig. 5.6 (B). Furthermore, nodes 1 and 4 are equidistant from 2 and 3, similarly nodes 6 and 9 from 7 and 8.

On the other hand, in Fig. 5.6(A), even though we still have the two groups related to their preference (participation in cascades), the arrangement of the nodes in each group captures their true closeness to one another and order of appearance in $\mathcal{C}$. For example, node 1 is closer to 4 than to 2 and 3, and node 4 is closer to 2 and 3 than 1 is to 2 and 3. We also see a similar trend in the members of g_2 according to their closeness recorded in $\mathcal{C}$. In addition, recall that nodes 1 and 8 have the tendency to trigger most of the diffusion events in the two groups as recorded by $\mathcal{C}$, and this fact can also be inferred from the embedding in Fig. 5.6 (A) and not (B). Therefore the former kind of embeddings can be exploited in problems like influence modeling.

5.4 Practical Consideration

For ease of implementation in both the delay-aware and delay-agnostic models, we have made the following assumptions:

1. Without loss of generality, we only consider the subset $V' \subseteq V$ of nodes that are observed in cascades (some nodes may never appear in cascades, and are thus ignored).

2. Each entry $\mathbf{e_u}[\mathbf{i}]$ of the event vector $\mathbf{e_u}$ is transformed to $\mathbf{e'_u}[\mathbf{i}] = \frac{1}{1+\mathbf{e_u}[\mathbf{i}]}$, where u is the index of a node, i is the index of a cascade, and $\mathbf{e_u}[\mathbf{i}] = t_j$ or $\mathbf{e_u}[\mathbf{i}] = j$ under the delay-aware and delay-agnostic settings, respectively. Given that

$$\lim_{\mathbf{e_u}[\mathbf{i}]\to 0} \frac{1}{1+\mathbf{e_u}[\mathbf{i}]} = 1 \quad and \quad \lim_{\mathbf{e_u}[\mathbf{i}]\to \infty} \frac{1}{1+\mathbf{e_u}[\mathbf{i}]} = 0, \tag{5.8}$$

each node u that occurs during the early stages of the i^{th} cascade will have $\mathbf{e'_u}[\mathbf{i}] \approx 1$; and each node v that have never occurred in C will have $\mathbf{e'_v}[\mathbf{i}] = 0$.

5.5 Problem Statement

After the definition of the above models, we are ready to formally define the network representation learning problem when G's topology is unknown as follows:

Problem 2. Given a model $\mathbf{M}$ that can be $f(\mathbf{P})$, $f(\mathbf{CF})$, $f(\mathbf{LF})$, $f(\mathbf{TF})$ or their concatenation $f(\mathbf{P}) \oplus f(\mathbf{CF}) \oplus f(\mathbf{LF}) \oplus f(\mathbf{TF})$, then the problem is to identify a function $\Phi : V \to \mathbb{R}^d$, where the optimization objective $\mathcal{L}$ is given by

$$\mathcal{L} = \min \sum_{\{u,v\} \subseteq V} ||\textsc{sim}(u, v; \mathbf{M}) - \textsc{sim}(u, v; \Phi)||_2^2 \tag{5.9}$$

and $f : \mathbf{M} \to \mathbb{R}^{n \times r}$ for $r \geq 1$ is a transformation function that maps $\mathbf{M}$ to a vector subspace, for example the identity function $f : \mathbf{M} \to \mathbf{M}$.

5.6 Unified Embedding

To make the computation more efficient, the NetTensor framework first pre-process the inputs; in particular, $\mathbf{P}$ and $\mathbf{CF}$ are transformed using TSVD, while the $\mathbf{LF}$ (local) and the $\mathbf{TF}$ (topic) features are passed as is through an identity function.

Recall that the matrix $\mathbf{P}$ is non-symmetric, that is, $\mathbf{P}[\mathbf{i}]$ and $\mathbf{P}[\mathbf{i}]^T$ are associated to node i's pairwise proximity by considering all infection events that happened *after* and *before* i's infection, respectively. Intuitively, we can think of $\mathbf{P}[\mathbf{i}]$ and $\mathbf{P}[\mathbf{i}]^T$ as models that capture node i's proximity when

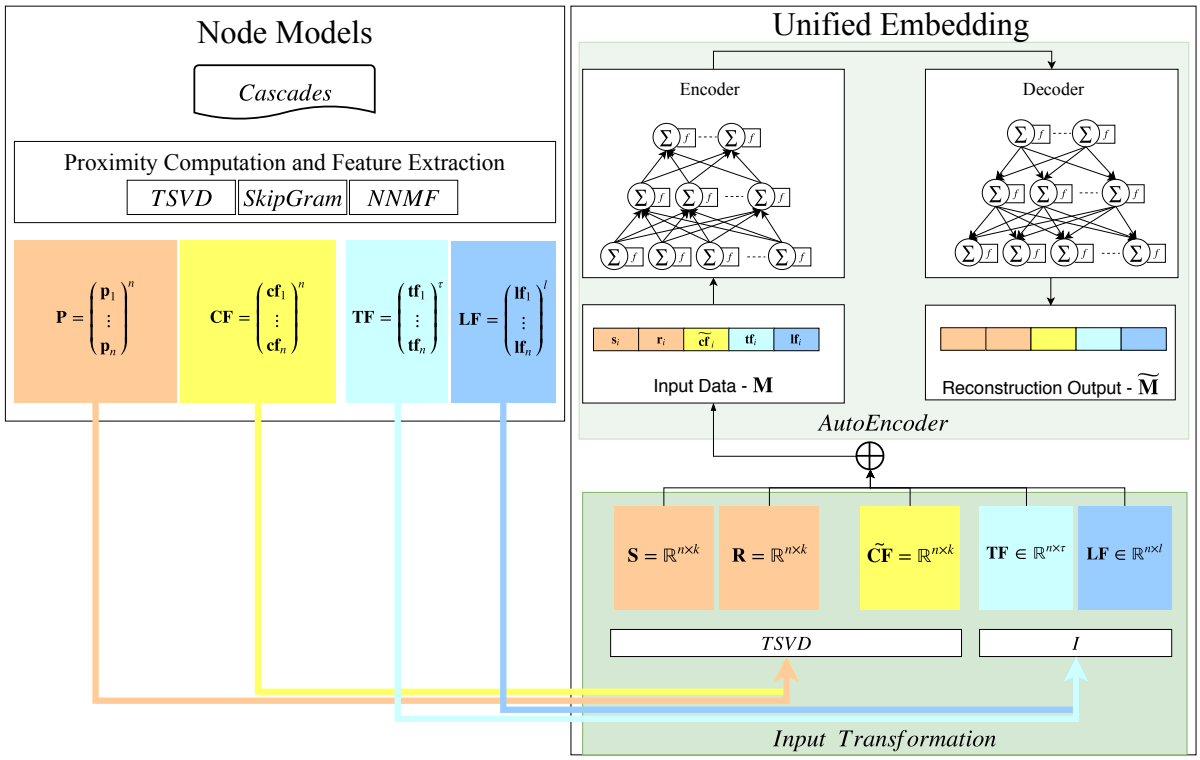


Figure 5.7: NETTENSOR Framework

sending and receiving contagions, respectively. Therefore we transform $\mathbf{P}$ into two matrices $\mathbf{S}, \mathbf{R} \in \mathbb{R}^{n \times k}$ using the $TSVD(\mathbf{P})$ function as:

$$\begin{aligned} \mathbf{P} &= \mathbf{A}\Sigma\mathbf{B}^T \\ \mathbf{S} &= [\sqrt{\sigma_i} \cdot \mathbf{A}[\mathbf{i}], \ldots, \sqrt{\sigma_k} \cdot \mathbf{A}[\mathbf{k}]] \\ \mathbf{R} &= [\sqrt{\sigma_i} \cdot \mathbf{B}[\mathbf{i}], \ldots, \sqrt{\sigma_k} \cdot \mathbf{B}[\mathbf{k}]] \end{aligned} \tag{5.10}$$

We apply a similar transformation on the statistical feature matrix $\mathbf{CF}$ – $TSVD(\mathbf{CF})$, however, since $\mathbf{CF}$ is a symmetric matrix, we materialize the transformation using the left singular vectors only and obtain

$$\tilde{\mathbf{CF}} = [\sqrt{\sigma_i} \cdot \mathbf{A}[\mathbf{i}], \ldots, \sqrt{\sigma_k} \cdot \mathbf{A}[\mathbf{k}]] \tag{5.11}$$

As of now, we have five dense matrices that encode proximity and different features of nodes, henceforth simply referred to as features. Our goal

is therefore to learn a unified embedding that preserves what is encoded in all these features.

A straightforward approach towards achieving this is simply to concatenate them. This however fails to capture the correlation between the different features, as each of them are trained separately. Therefore, we learn a unified embedding that enables us to optimize the embedding model parameters simultaneously as suggested in [55].

For this purpose, we propose to use a simple and unsupervised neural network model, an *autoencoder*. As shown in the 'Unified Embedding' component of Fig. 5.7, an auto-encoder has two components called *encoder* and *decoder*.

Intuitively, the encoder generates a compressed code $\Phi[\mathbf{i}]$ representing the unified embedding from an input vector $\mathbf{M}[\mathbf{i}]$; the decoder attempts to reconstruct, $\tilde{\mathbf{M}}[\mathbf{i}] \approx \mathbf{M}[\mathbf{i}]$, the input vector from the compressed code. Let

$$\mathbf{M} = \mathit{TSVD}(\mathbf{P}) \oplus \mathit{TSVD}(\mathbf{CF}) \oplus I(\mathbf{TF}) \oplus I(\mathbf{LF}) = \mathbf{S} \oplus \mathbf{R} \oplus \tilde{\mathbf{CF}} \oplus \mathbf{TF} \oplus \mathbf{LF}$$

then, the objective of the auto-encoder is normally specified as minimizing the squared distance as

$$\begin{aligned} &\min ||\mathbf{M} - \tilde{\mathbf{M}}||_F^2 \\ &\Phi = \textsc{enc}^L(\ldots(\textsc{enc}^{\mathbf{1}}(\mathbf{H}^{\mathbf{1}}_{\mathbf{enc}} \cdot \mathbf{M}))\ldots) \\ &\tilde{\mathbf{M}} = \textsc{dec}^L(\ldots(\textsc{dec}^{\mathbf{1}}(\mathbf{H}^{\mathbf{1}}_{\mathbf{dec}} \cdot \Phi))\ldots) \end{aligned} \tag{5.12}$$

where $\textsc{enc}^{\mathbf{i}}$ and $\textsc{dec}^{\mathbf{i}}$ are the activation functions, such as `tanh` and `relu`, and $\mathbf{H}^{\mathbf{i}}_{\mathbf{enc}}$ and $\mathbf{H}^{\mathbf{i}}_{\mathbf{enc}}$ are weight matrices of the encoder and decoder, respectively, at the i^{th} layer and L is the total number of layers.

The optimization objective of Eq. 5.12 can then be solved using the stochastic gradient descent method or its variants. Once the optimization has converged, after all the model parameters $\mathbf{H}^{\mathbf{i}}_{\mathbf{enc}}, \mathbf{H}^{\mathbf{i}}_{\mathbf{dec}} : i = 1, \ldots, L$ have been fixed to an optimal value, we take $\Phi = \textsc{enc}^L(\ldots(\mathbf{H}^{\mathbf{1}}_{\mathbf{enc}} \cdot \mathbf{M})\ldots) \in$

$\mathbb{R}^{n \times d}$ as the unified embedding. Furthermore, during the training phase we apply a dropout regularization to avoid over-fitting.

5.7 Experimental Evaluation

In this section, we compare the performance of our approach with strong and popular baselines for network representation learning, against five real datasets.

It is important to stress again that the baselines require the complete knowledge of the structural information of the network, while our results are obtained by only using information about cascades, without any knowledge of the network at all. Our goal is to achieve performance levels as close as possible to the state-of-the-art NRL techniques, but we are perfectly satisfied by results that are marginally lower than baselines.

The baselines are trained using the source code provided directly from authors. For our algorithm, we separately report the results of NetTensor$_{time}$ and NetTensor$_{order}$, i.e. the variants of NetTensor corresponding to delay-aware and delay-agnostic approaches, respectively.

5.7.1 Datasets

The evaluation is carried out on the following datasets, summarized in Table 5.1.

1. Memetracker [52]: The same Memetracker dataset described in the previous chapter.

2. Twitter [82, 83]: Two datasets associated with users and their interactions through hashtags and retweets on the Twitter social network. There are two kinds of cascades constructed using retweet and hashtag activities of users. The underlying network contains reciprocal

Dataset	**#Nodes**	**# of Edges**	**#Cascades**
Memetracker	259,136	7,765,325	71,568
Twitter Hashtag (HT)	500,294	11,781,154	128,611
Twitter Retweet (RT)	292,061	6,613,937	23,703
Yelp	175,305	2,166,226	53,312
MovieLens	6040	9,985,110	3,592

Table 5.1: Dataset summary.

Feature	Weighted-L1	Weighted-L2	Hadamard	Average
$\Phi[uv, i]$	$\lvert\Phi[u, i] - \Phi[v, i]\rvert$	$\lvert\Phi[u, i] - \Phi[v, i]\rvert^2$	$\Phi[u, i] * \Phi[v, i]$	$\frac{\Phi[u,i]+\Phi[v,i]}{2}$

Table 5.2: Edge feature construction techniques. $\Phi[uv]$ is an edge feature vector for a pair of nodes $u, v \in V$ and $\Phi[uv, i]$ is the i^{th} component.

follower links.

3. Yelp [1]: A review dataset containing user reviews about businesses. A cascade is associated with a sequence of reviews given to a business overtime. Reviews up to 2014 are used. The underlying network contains friendship links.

4. MovieLens 1M [31]: A movie review dataset containing user reviews on movies. Cascades are generated in the same way as Yelp. The dataset however does not have an actual interaction network, therefore we build a co-rating network of users. That is, we add an edge between a pair of users u and v, if they review a number τ of common movies. We simply set $\tau = 22$, which is the mean value of the number of co-rated movies, that generates a dense network with $\approx 10M$ edges.

For all the datasets the ground truth network contains users that are involved in the cascades and this will enable us to have a fair comparison between NetTensor and the baselines.

[1] https://www.yelp.com/dataset

5.7.2 Baselines

The research in NRL has recently produced a plethora of studies focusing on different properties of networks. However, it is not possible to provide an exhaustive comparison with all the existing methods, and hence we pick three among the most popular state-of-the-art NRL techniques, i.e., DeepWalk [64], Node2Vec [28] and Line [70]. A comprehensive overview of these algorithms is given in Chapter 7.

5.7.3 Link Prediction

In Chapter 4, we have already established that link prediction is one of the applications where NRL techniques play a crucial role, and here we present the experimental evaluation results on this task.

Setting

We adopt here the same strategy followed by previous studies [28, 77, 22], namely we remove a certain percentage, denoted *rate*, of the edges from the ground truth network, while ensuring that the residual network remains connected as suggested in [28]. The removed edges will be considered as true edge samples, while an equal number of false edges (edges that do not exist in the network) will be sampled for comparison.

During the NRL phase, we train the baseline methods on the residual network; given that our method requires no knowledge of the network structure, we simply train it on the cascades. Finally, in line with existing studies [28, 22], we learn features for the sampled edges using the four techniques listed in Table 5.2. All the results are reported as percentages and higher means better.

The hyper-parameters are tuned using random search and the final configuration of NetTensor is as follows: $\mathbf{S}, \mathbf{R}, \tilde{\mathbf{CF}} \in \mathbb{R}^{n \times 256}$ and $\mathbf{TF}, \mathbf{LF} \in$

$\mathbb{R}^{n\times 128}$. This leads to a combined node model $\mathbf{M} \in \mathbb{R}^{n\times 1024}$. The size of the layers of the autoencoder in the unified embedding model is then $[1024, 512, 256, 128]$, and hence $\Phi \in \mathbb{R}^{n\times 128}$, with a dropout regularization rate of 0.4. To solve the optimization objective of Eq. 5.12 we use the Adam stochastic optimization technique [43] with a learning rate of 0.0001. The proximity window size to compute $\mathbf{S}$ and $\mathbf{R}$ is set to 100 for all datasets but HT, for which it is set to 50. The context window size to compute $\mathbf{LF}^l$ is set to 15 for all datasets. For all the baselines, the nodes embedding size is set to 128; for Node2Vec, parameters p and q are set to 1. For DeepWalk and Node2Vec, the number of walks and walk length are set to 10 and 80, respectively. All the results are reported as percentages and higher means better.

Results and Discussions

In Table 5.3 and 5.4, we report the Area Under Curve (AUC) results, with $rate = 30\%$ and $rate = 50\%$, using the edge feature construction techniques listed in Table 5.2. In almost all the cases, the *Average* technique gives the best results, underlined in the table; all the baselines score their lowest with Hadamard. The bold results show the algorithm with the best performance. As it can be seen from Table 5.3, NetTensor achieves a good result that is marginally smaller (for HT, RT and Yelp) and marginally greater (for Memetracker) than the baselines, for $rate = 30\%$.

In Table 5.4 we report the results with $rate = 50\%$; NetTensor achieves the best results with Memetracker and RT, while Node2Vec and Line are the best algorithm for HT and Yelp, respectively.

Note that even though NetTensor gives us the flexibility to choose between different models, we have used here the full model that includes all the feature types. As described in Section 5.7.6, however, we can achieve a very good performance even using simplified models.

Algorithm	**Feature**	***Dataset***			
		Memetracker	**HT**	**RT**	**Yelp**
NetTensor$_{time}$	Weighted-L1	92.74	86.41	85.39	92.14
	Weighted-L2	93.02	83.48	82.29	92.63
	Hadamard	90.67	66.86	66.17	77.21
	Average	**98.21**	96.30	95.74	97.16
NetTensor$_{order}$	Weighted-L1	92.87	82.85	85.92	91.73
	Weighted-L2	93.21	80.27	81.90	92.21
	Hadamard	92.28	68.72	66.32	81.12
	Average	97.76	95.68	92.56	97.27
DeepWalk	Weighted-L1	92.15	82.10	84.34	99.86
	Weighted-L2	92.35	80.53	82.38	99.84
	Hadamard	90.96	84.20	80.42	97.24
	Average	87.03	95.79	96.19	99.63
Line	Weighted-L1	84.92	86.01	85.06	99.01
	Weighted-L2	85.81	89.15	87.99	98.54
	Hadamard	80.14	79.13	81.73	97.01
	Average	96.97	97.52	97.85	**99.89**
Node2Vec	Weighted-L1	85.64	83.53	82.56	99.34
	Weighted-L2	85.87	81.98	82.04	99.45
	Hadamard	82.34	71.36	68.51	88.60
	Average	91.42	**97.66**	**97.86**	99.85

Table 5.3: AUC score for link prediction with *rate* = 30%. Bold indicates the best performing algorithm for a dataset and underline indicates the best performing feature construction technique for each dataset and each algorithm.

5.7.4 Network Reconstruction

One way to measure the quality of NRL algorithms is their ability to reconstruct the original graph [77]. Besides, network reconstruction from diffusion cascades is an important research question in the area of social network analysis. For these reasons, we have evaluated the performance of our algorithm and the baselines in reconstructing the original network structure.

Algorithms	Datasets			
	Memetracker	**HT**	**RT**	**Yelp**
NetTensor$_{time}$	**95.84**	92.31	94.00	96.49
NetTensor$_{order}$	95.42	92.72	**95.02**	95.92
DeepWalk	84.91	94.46	92.92	96.67
Line	86.92	95.13	94.20	**99.89**
Node2Vec	92.99	**95.18**	94.13	99.46

Table 5.4: AUC score for link prediction, with *rate* = 50% and *Average* edge feature learning method. Bold indicates the best performing algorithm for a dataset.

Setting

For this experiment, the baselines are trained using the complete network structure, while NetTensor is trained using the cascades. In line with existing studies [77, 22], we adopt the precision-at-K ($P@K$) metric, which is already defined in Chapter 4 – Eq. 4.8. Recall that the $P@K$ metric is based on a $rank(u, v)$ function according to the score on the edges. Here we seek to compute score for pairs u, v and we use the sigmoid as a scoring function $score(u, v)$:

$$score(u, v) = \frac{1}{1 + e^{-dot(\Phi_u, \Phi_v)}}$$

where $dot(\mathbf{x}, \mathbf{y})$ is the dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$.

Computing the $score(\cdot, \cdot)$ function for all pairs of nodes is expensive – $O(|V|^2)$ – for large networks. Instead, we sample pairs including all the true edges and a factor λ of pairs that are not connected, drawing in total $|E| + \lambda \times |E|$ sample pairs out of all $|V|^2$ pairs of nodes.

The same configuration of hyper-parameters as in the link prediction task is used.

Algorithm	K	Dataset			
		Memetracker	**HT**	**RT**	**Yelp**
NetTensor$_{time}$	100K	97.01	99.94	98.96	93.85
	500K	87.54	99.67	94.33	70.97
	1M	81.67	99.36	81.93	62.34
NetTensor$_{order}$	100K	99.11	99.95	99.03	93.76
	500K	90.06	99.68	94.35	70.37
	1M	83.10	99.34	81.58	62.02
DeepWalk	100K	**100**	**100**	**100**	**100**
	500K	**99.99**	**100**	**99.88**	69.69
	1M	**99.98**	**100**	99.79	62.36
Line	100K	**100**	62.84	68.21	55.14
	500K	67.31	62.85	62.69	53.00
	1M	60.89	62.29	60.15	51.66
Node2Vec	100K	**100**	**100**	99.95	99.82
	500K	**99.99**	**100**	99.86	**90.27**
	1M	99.97	**100**	**99.80**	**78.14**

Table 5.5: P@K results for the network reconstruction task, $\lambda = 1$.

Algorithm	λ	Dataset			
		Memetracker	**HT**	**RT**	**Yelp**
NetTensor$_{time}$	2	79.99	99.21	90.00	58.53
	3	74.59	98.77	85.90	51.55
NetTensor$_{order}$	2	83.46	99.23	89.79	58.01
	3	78.56	98.75	85.58	51.15
DeepWalk	2	**99.36**	**99.99**	**99.36**	33.26
	3	**99.98**	**99.99**	**98.71**	4e-6
Line	2	58.84	47.74	50.44	37.63
	3	54.70	39.16	42.77	30.59
Node2Vec	2	**99.29**	**99.99**	99.29	**86.25**
	3	**99.97**	**99.99**	98.62	**83.95**

Table 5.6: P@K results for the network reconstruction task, $K = 500K$.

Results and Discussion

Table 5.5 and 5.6 show the results of the network reconstruction task for fixed $\lambda = 1$ and $K = 500K$, respectively. For all the datasets, either one

or both the variants of NetTensor achieve a reasonably close result with respect to that of Node2Vec and DeepWalk, with a gap between 0.06% and $\approx 18\%$ for values of K ranging from 100K to 1M. It is interesting to note that both variants perform significantly better than Line, except in one case (Memetracker, $\lambda = 1$ and $K = 100K$). As we introduce more noise, which is controlled by λ, a significant decrease for all the algorithms is observed for the Yelp dataset. NetTensor performs significantly worst than Node2Vec (a difference around 32%), while it performs better than DeepWalk (a decrease around 4%) and Line.

Another important observation is that the difference between the two variants of our algorithm are insignificant, except for the $\approx 4\%$ difference in the Memetracker dataset, as shown in Table 5.6. A plausible explanation is that in the other datasets, the nodes interact in a single OSN platform that enables them to be aware of their neighbors activity, for example using the news feed of Twitter. In such condition, it is likely that time and order capture the interaction patterns equally. For Memetracker, on the other hand, interaction occurs across different platforms, such as *bbc.com* and *aljazeera.com*, requiring users in Memetracker to intentionally browse other node's page to follow their activities. In such cases, the order of how information propagates could uncover a better interaction pattern than reaction times; a similar argument is raised by Du et.al. [18].

5.7.5 Node Classification

The last but not the least application that we consider is node classification. We evaluate the algorithms performance using the MovieLens dataset, as it is the only one with labels.

Setting

The dataset contains three kinds of labels, which are *age* (7 classes), *occupation* (21 classes) and *gender* (binary classification, male/female).

As there is no underlying interaction/influence network that lead to the cascades that we have extracted, NetTensor simply uses the statistical and local-context features in this experiment. As discussed in Sections 5.3.1 and 5.3.2, these features are oblivious to time and order and simply capture contextual patterns.

In all the experiments, we perform a K-fold cross validation by using a certain fraction TR of the data to train the model, while the rest is used to test the trained model. We use $K = 10$ in all the experiments and report the expected value along with the standard deviation at 95% confidence interval. All the results are reported as percentages and higher means better. A similar configuration of hyper-parameters as in the previous experiments is used. As NetTensor uses only two of its features, however, the layers of the autoencoder in the unified embedding model are configured as $[384, 200, 128]$. Similar to existing NRL studies, we use macro-F1 and micro-F1 evaluation metrics.

Results and Discussions

The Micro-F1 and Macro-F1 classification results over the three types of labels are reported in Tables 5.7 and 5.8. Overall, NetTensor significantly outperforms all the baselines in both metrics. It only achieve a slightly lower Micro-F1 result than Node2Vec for the occupation classification task. Alhough it is difficult to establish that NetTensor is superior than the baselines in the above task, as they are not trained on an actual interaction network, it can however shade light on the strong potential use of models trained using cascades for node classification.

Label	TR	Algorithms			
		NetTensor	DeepWalk	Line	Node2Vec
Age	10%	**41.35 ± 0.01**	35.45 ± 0.007	36.39 ± 0.002	36.28 ± 0.002
	30%	**44.97 ± 0.009**	36.51 ± 0.005	36.46 ± 0.003	36.37 ± 0.006
	50%	**46.25 ± 0.006**	36.85 ± 0.004	36.36 ± 0.006	36.59 ± 0.005
Gender	10%	**78.35 ± 0.006**	75.42 ± 0.003	73.83 ± 0.001	73.84 ± 0.002
	30%	**80.11 ± 0.004**	75.87 ± 0.003	73.88 ± 0.004	73.87 ± 0.004
	50%	**80.55 ± 0.007**	76.17 ± 0.006	74.02 ± 0.006	74.00 ± 0.006
Occupation	10%	16.00 ± 0.008	10.87 ± 0.006	11.78 ± 0.008	**17.36 ± 0.007**
	30%	17.62 ± 0.005	10.74 ± 0.004	12.13 ± 0.010	**18.71 ± 0.004**
	50%	18.81 ± 0.005	10.86 ± 0.004	12.57 ± 0.006	**18.90 ± 0.005**

Table 5.7: Micro-F1 results for node classification alongside the standard deviations with 95% confidence interval.

Label	TR	Algorithms			
		NetTensor	DeepWalk	Line	Node2Vec
Age	10%	**21.18 ± 0.018**	14.37 ± 0.01	7.80 ± 0.001	8.13 ± 0.006
	30%	**26.89 ± 0.018**	17.13 ± 0.008	7.84 ± 0.001	8.11 ± 0.006
	50%	**29.53 ± 0.010**	17.59 ± 0.004	7.87 ± 0.001	8.24 ± 0.006
Gender	10%	**64.48 ± 0.018**	55.01 ± 0.018	42.54 ± 0.001	42.47 ± 0.0006
	30%	**69.65 ± 0.013**	59.49 ± 0.012	42.59 ± 0.001	42.48 ± 0.001
	50%	**70.92 ± 0.009**	60.65 ± 0.012	42.79 ± 0.002	42.52 ± 0.002
Occupation	10%	**5.59 ± 0.005**	4.24 ± 0.004	1.82 ± 0.002	2.48 ± 0.004
	30%	**7.26 ± 0.004**	5.06 ± 0.002	1.90 ± 0.004	2.65 ± 0.002
	50%	**8.04 ± 0.004**	5.31 ± 0.004	1.99 ± 0.002	2.85 ± 0.002

Table 5.8: Macro-F1 results for node classification alongside the standard deviations with 95% confidence interval.

5.7.6 Node Model Analysis

In this section, we analyze the effect of each node model (feature) independently on the tasks that we have carried out in the previous sections. In the following set of experiments, we use Twitter RT and Yelp for link prediction and network reconstruction, given their relatively small size; instead, we use MovieLens for the node classification task.

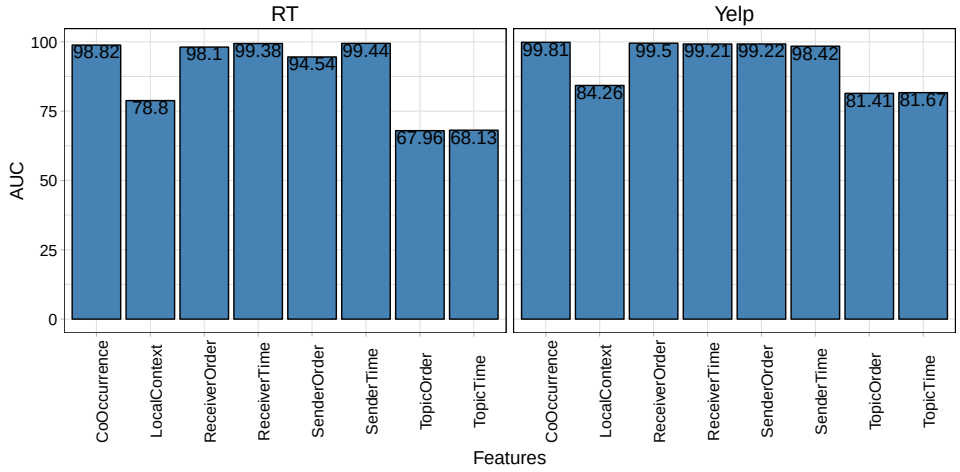


Figure 5.8: Performance of Node Models in Link Prediction.

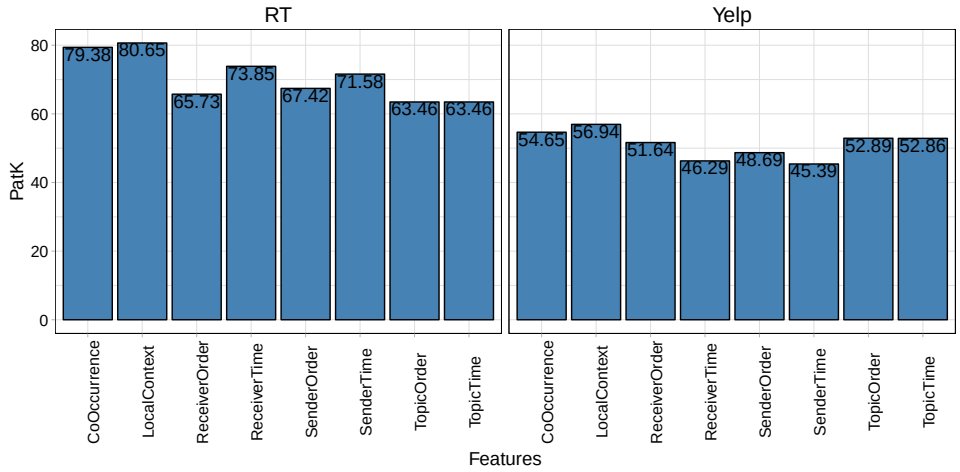


Figure 5.9: Performance of Node Models in Network Reconstruction, $\lambda = 2$ and $K = 500K$.

We start our analysis with link prediction, and in Fig. 5.8 we report the results we have obtained for all the features. As depicted in the figure, topic is the least informative feature for link prediction in both datasets. Local context features are the second least informative ones and the rest of them are more or less similar and strongly informative for this task.

Another observation is that the delay-aware features achieve a higher performance compared to the delay-agnostic variants in the Twitter RT

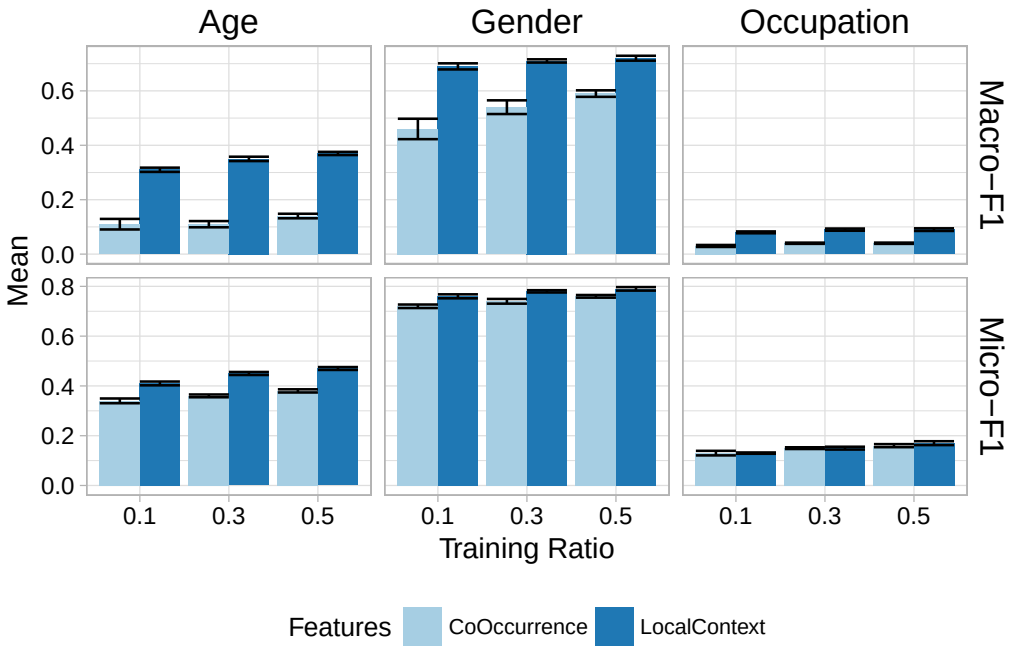


Figure 5.10: Performance of the two kinds of features in the three types of node classification tasks.

dataset. The difference is particularly pronounced for the sender features, that is, 94% and 99%. We conjecture that the existence of a dedicated platform where users can be notified of their friends activity has contributed to such manifestation of a stronger interaction patterns of reaction time than order.

Next, we move to the analysis of the features for the network reconstruction task. The results are reported in Fig. 5.9. Similarly to the previous experiment, we observe that delay-aware models perform better in the RT dataset, adding more evidence to our conjecture. In addition, we observe that the statistical (co-occurrence) and local context features also give a comparable result for this task.

Finally, we analyze the performance of the statistical (co-occurrence) and local context features used in the node classification task. Fig. 5.10 reports their performance in this task and shows that local-context features

are more predictive than the statistical ones.

5.7.7 Parameter Analysis

NetTensor has a few hyper-parameters that needs to be tuned:

- In the proximity model, the window size pw
- In the local-context features, the context window size cw; if the optional sequence sampling is carried out during the local-context feature construction, the number ns of sequences per node and the sequence length sl should be added as well;
- The dimension sizes of the features $\mathbf{TF} \in \mathbb{R}^{n \times \tau}$, $\mathbf{LF} \in \mathbb{R}^{n \times l}$, i.e. τ, l respectively
- Finally, the configuration of the unified embedding model, i.e., the number of layers L and the number of neurons (units - nu) in each layer.

In total, NetTensor has thus either 7 or 9 hyper-parameters depending on the use of the optional sequence sampling step of the local feature extraction in Section 5.3.2. Such large number of hyper-parameters is due to the multiple components of NetTensor. As shown below, however, our algorithm is sensitive only to a very few of them; besides, as illustrated in the previous subsection, NetTensor gives us the flexibility to use one or more of the individual components based on the task on hand.

Fig. 5.11 shows the effect of proximity window, context window and embedding size on the link prediction and network reconstruction tasks. The embedding size is the one of the unified embedding model. As shown in the figures, NetTensor is not sensitive to the variation of these parameters.

For the node classification task, we have only analyzed the effect of the embedding size as we have discarded the proximity and node features for

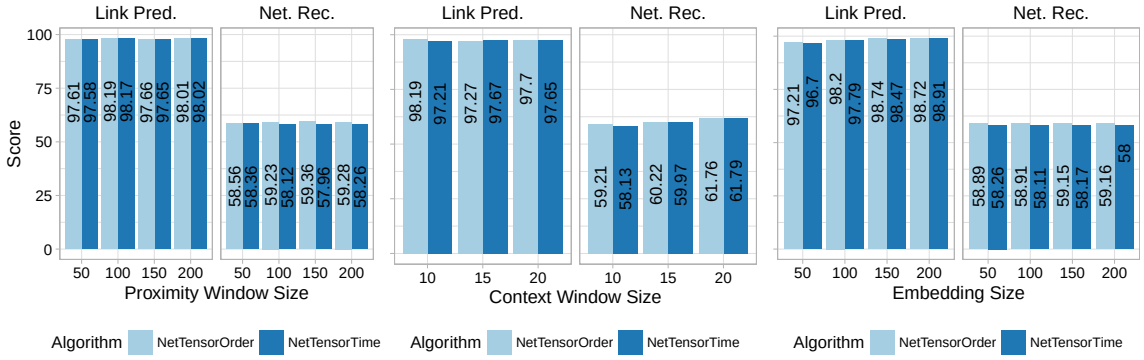


Figure 5.11: Effect of proximity window, local context window, and embedding sizes, AUC and $P@K$ are the scores for link prediction (Link Pred.) and network reconstruction (Net. Rec.), respectively.

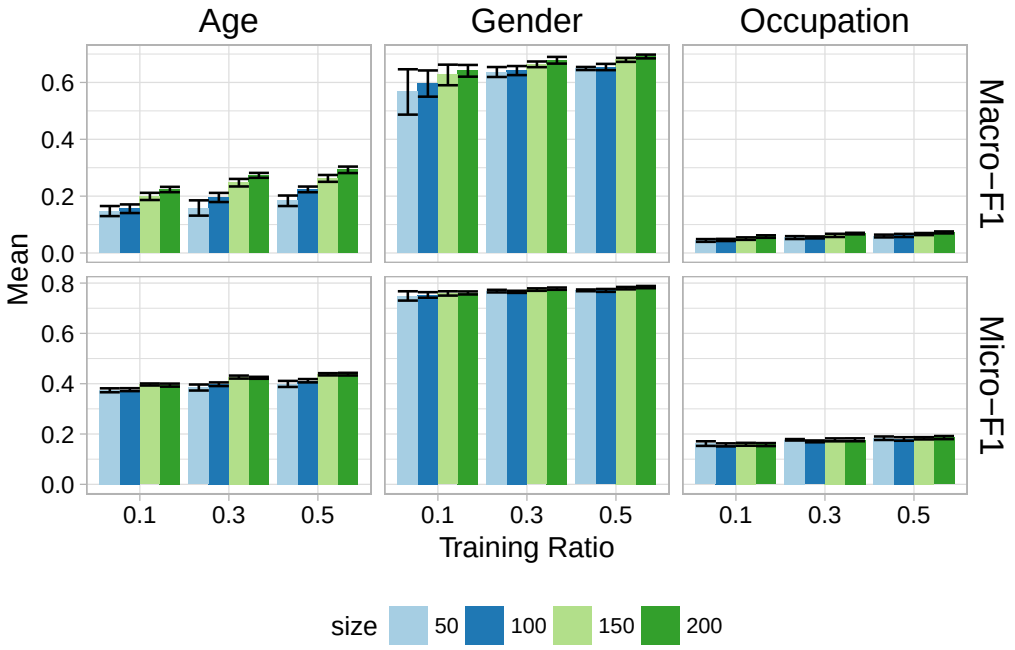


Figure 5.12: Effect of the final embedding size d on node classification

the reasons discussed in Section 5.7.5. Fig. 5.12 shows the effect of this parameter. As expected, improvements can be seen with the increase of the embedding size; this is particularly pronounced for Macro-F1.

5.7.8 Application to Learning Influence Propagation Probabilities

Influence maximization is an important area of research in social network analysis, with relevant real-world applications such as the identification of the top-k influencers in the network. Motivated by this need, the seminal work of [41] has introduced two approaches, the independent cascade model (IC) and the linear threshold model, to tackle such problem. For illustration purpose, we focus on the former.

According to the IC model, a pair of users u, v has a probability $P(u, v)$ of spreading influence among each other. In the iterative influence maximization procedure, at every step each user u influenced in the previous step will be given one chance of spreading influence to its uninfluenced neighbors v with a probability $P(u, v)$.

Several studies have been suggested to learn the aforementioned probabilities from cascades [58, 25, 26, 27, 9, 38].

However, some of them discard the reaction time or the order of nodes infection in these events; some assume that the propagation probability is symmetric, i.e. $p(u, v) = p(v, u)$; others assume a fixed parametric form of influence propagation rates, such as exponential or power-law distributions. A study, however, has empirically shown that these are strong assumptions and real networks exhibit a more complex dynamics [18].

We propose here a simple data-driven technique that uses the sender and receiver node proximity models to estimate these probabilities. Our scheme is non-symmetric and it is based on the behavior of users or their tendency in spreading and receiving influence. As discussed in Section 5.2, the node-proximity models of the sender $\mathbf{S}$ and the receiver $\mathbf{R}$ are designed in such a way that they capture the pattern in users tendency to spreading and receiving influence. Thus, we assume that the probability of a node

u influencing another node v depends on node u's tendency to spread influence and v's tendency to receive influence; in other words, $\mathbf{S}[\mathbf{u}]$ and $\mathbf{R}[\mathbf{v}]$, respectively.

Therefore, we specify a unified framework to estimate the influence propagation probabilities between a pair of users as:

$$p(u, v) = \frac{1}{1 + e^{-dot(\mathbf{S}[\mathbf{u}], \mathbf{R}[\mathbf{v}])}} \tag{5.13}$$

Note that $p(u, v) = p(v, u)$ does not necessarily hold. Furthermore, one has the flexibility to use the delay-aware or delay-agnostic versions of $\mathbf{S}[\mathbf{u}]$ and $\mathbf{R}[\mathbf{v}]$, if needed.

Chapter 6

Cascade Representation Learning for Virality Prediction

In the previous chapters, we have observed how we can use diffusion events in order to aid NRL when the structure of the network is known, partially known, or hidden. Obviously, the output of NRL algorithms is a low-dimensional, dense embedding of nodes in an information network. We have seen how these embeddings can be used for different kinds of network analysis problems. Unlike traditional techniques that are based on manual feature extractions, these embeddings are not task-dependent: the same embedding of nodes is used across different problems.

One important problem where we might apply such embeddings is predicting the future states of diffusion events themselves. Online social networks are the quintessence of information networks where diffusion events take place, in the form of posts or tweets that start from a few sources and then suddenly spread like a wildfire. Just to mention a recent example, the post celebrating the landing of the Falcon-Heavy rocket sent from the SpaceX[1] Twitter account on February 6$^{\text{th}}$, 2018, has been retweeted more than 75k times within the same day of posting. Such diffusion events are called *viral cascades* and predicting them at early stages is vital for different

[1] https://twitter.com/SpaceX

applications, for example to forecast trends and rumor break-outs [97].

As pointed out in the previous chapter, it is common to be in a strenuous situation to have access to the network structure where the diffusion events occur. For this reason, recent studies [68, 97] have dedicated several efforts to the prediction of content popularity with the focus of achieving good predictions in the shortest possible time, using as small information as possible about the underlying network structure.

Early research on predicting cascade virality assumed strong correlations between the propagation of content and the structural properties of early starters of the spreading events. Therefore, most of the early attempts towards predicting the virality of cascades have relied on manually extracted features from the underlying network structure and the cascade itself [53, 82, 94, 93, 12, 69]. Information such as the number of followers/followees that engaged users have, users connections and community structure, activity level, etc., have been exploited.

This, however, poses two kinds of issues. First, manual feature crafting is an expensive and challenging task. In most cases, domain knowledge and external information about the content in question is required. For instance, content popularity may be linked to several parameters, such as event topic, external events or the content relevance to given periods of time (e.g., posting about football during the World Cup), etc.

Besides, the optimal number and relevance of features that need to be extracted is not obvious, making it difficult to decide when to stop looking for additional ones [28]. Furthermore, some recent cascade examples show different spread patterns even when showing similar network properties of the engaged nodes in the underlying social graph; thus, network properties may not be the optimal or the only indicator for virality. For example, Fig. 6.1(A) shows the spread patterns of two hashtag campaigns (`#metoo`

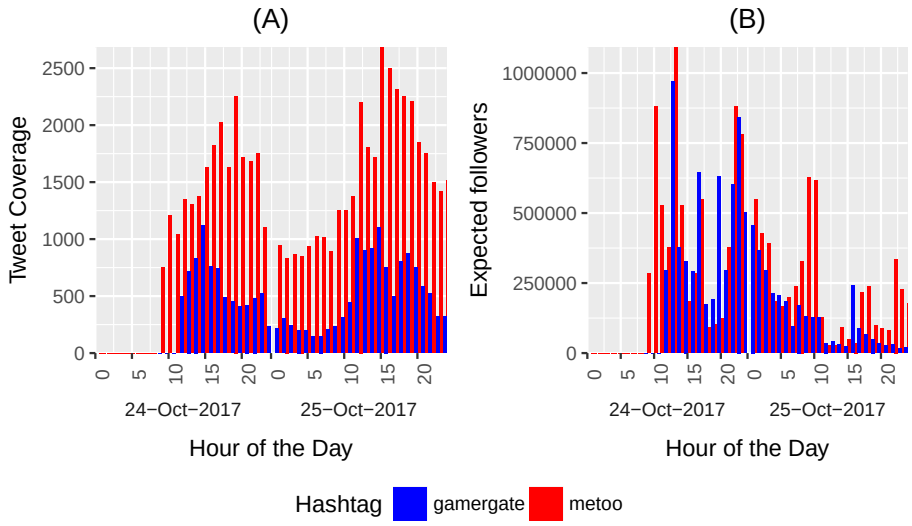


Figure 6.1: Examples of two recent hashtag campaigns. (A) The tweeting frequency of each hashtag; `#metoo` achieved more spread compared to `#gamergate`. (B) The network properties of the participating nodes in each hashtag in terms of average number of followers; the nodes engaged in the first 12 hours almost achieve similar reachability in both hashtags.

and `#gamergate`) that happened almost at the same time[2]. As shown, `#metoo` went viral in the first two days. The hashtag `#metoo` was tweeted more than 200k times by the end of October 15, 2017[3]. On the other hand, `#gamergate` did not become viral like `#metoo`, even though they have reasonably similar network properties such as the expected number of followers (indicator for potential spread in the future) of early starters, as shown in Fig. 6.1(B).

In addition to that, acquiring information about the social network structure is usually very expensive for those who work outside the companies hosting the data. For example, for popular social networks such as Twitter and Facebook, it may take several months to extract just a portion

[2]The dataset for these two hashtags is collected based on information available via `https://github.com/datacamp/datacamp-metoo-analysis` and `https://github.com/awesomedata/awesome-public-datasets`, respectively

[3]https://en.wikipedia.org/wiki/Me_Too_movement

of the network. Moreover, due to privacy constraints and policies of such systems, the extracted network is usually lacking a significant amount of structural information, such as edges of some users participating in hashtag campaigns who set their connections to be private [65].

For the reasons above, it becomes imperative to design algorithms that do not require manual feature engineering or information about the underlying network, but are still capable of effectively predicting cascade virality in the very early stages of the diffusion. Some initial but also strong attempts towards exploring this *network-agnostic* approach have already demonstrated the potential for effective and timely prediction, merely based on information that could be extracted from the cascades themselves without requiring any other additional information [68]. However, most of the works available in the literature are mainly adopting either *"network-aware"* or at best *"quasi-network-agnostic"* approaches [97], relying on "less expensive" structural information, such as node degrees.

In this chapter we propose a novel *network-agnostic* algorithm called CAS2VEC that predicts cascade virality simply based on information explicitly available in the cascade itself (i.e., *the time between share events*). Our main premise is that the reaction time between the sequence of events encoded in a cascade is often a sufficient indicator to whether it will become viral or not in the near future. The reaction times in the early sequence of events can be used to model the cascade initial speed (i.e., the speed by which a cascade starts its spread), as well as its momentum.

By analyzing the distribution of reaction times for viral and non-viral cascades on multiple datasets, and based on corroborating observations supporting our premise, we have modeled cascades as a series of timestamps, where each element of the series is the reaction time measured from the source signal. Furthermore, our work is partly inspired by iSAX [10], that is used for indexing time series data. Particularly, we apply a similar

technique as iSAX on cascades to transform them into instances of one-dimensional point processes in time space, such that each point of the time series of the cascade is a discrete value obtained by using equally-sized periods of times.

Finally we automatically learn representations of cascades that are capable of predicting virality using the transformed cascades.

6.1 Summary of Contributions

CAS2VEC provides a novel network-agnostic approach that models information cascades as time series by discretizing them using time slices. Essentially, CAS2VEC learns high-quality features that can predict whether a cascade is going to become viral or not, simply by exploiting time series data computed from the cascades. To show the effectiveness of the learned representations in cascade prediction, we have performed extensive experiments and compared it against strong baselines.

Our results show that in predicting virality, the features learned using CAS2VEC outperform the baselines by more than an order of magnitude. Besides being able to perform predictions, its important to make them as early as possible. Thus we have shown that, compared to the state-of-the-art, the CAS2VEC performance particularly stands-out in detecting the virality of a popular content at the very early stage of its growth.

6.2 Background and Problem

The cascade definition that we introduced in Chapter 2 models a series of share events associated with the infection of users. Given that we are adopting a network-agnostic approach, we shall have no assumptions regarding the identity and underlying connectivity of users, and we will sim-

ply consider a cascade as a sequence of events.

For this reason, we strip out any user information and re-formulate cascades as a series of timestamps:

$$C = [t_1, \ldots, t_{|C|}] \tag{6.1}$$

We use $trace(C, t_b, t_e)$ to denote the sub-sequence of events whose timestamps are between the beginning time t_b (included) and the end time t_e (excluded):

$$trace(C, t_b, t_e) = [t : t \in C \wedge t_b \leq t < t_e] \tag{6.2}$$

For the sake of brevity, we use $trace(C, t_e)$ to denote the prefix of the subsequence including the events occurring before t_e since the initial event $t_1 = 0$, i.e.

$$trace(C, t_e) = trace(C, t_1, t_e) \tag{6.3}$$

The features that we intend to learn for cascades could be optimized for different types of prediction problems. In this chapter, we will focus learning features that are optimized for *virality prediction*, i.e. the task of deciding whether a cascade, after an observation period, is going viral or not before a given amount of time. From a practical perspective, this is a very important challenge [68].

To formally state our problem, we first define an *observation* of C

$$trace(C, t_o) = [t_1 = 0, \ldots, t_i = t_o]$$

for the early hours starting from the beginning $t_1 = 0$ of the cascade C up to an *observation time* t_o. We refer to the time period between 0 and t_o as an *observation window*.

Given an observation $O = trace(C, t_o)$, we then define a *prediction window* or a *delay window* Δ as a period starting from time t_o and up to $t_o + \Delta$ time, after which we want to establish whether a specific cascade

C is going viral or not. In other words, at a *prediction time* $t_p = t_o + \Delta$, our goal is to examine the state $trace(C, t_p)$, which is by comparing it size $|trace(C, t_p)|$ against a threshold.

Similar to existing studies [82, 68], we consider two ways of choosing a threshold that governs whether a cascade is viral or not:

- through an absolute threshold $\theta_a \in \mathbb{R}^+$, the cascade C is viral if $|trace(C, t_p)| \geq \theta_a$;
- through a relative threshold $\theta_r \in (0, 1)$, the cascade C is viral if $|trace(C, t_p)| > |C'|, \forall C' \in \mathcal{C} \setminus perc(\mathcal{C}, \theta_r)$, where $perc(\mathcal{C}, \theta_r)$ is the θ_r-percentile set of largest cascades among the cascades in $\mathcal{C}$ and it can be formally specified as

  $$perc(\mathcal{C}, \theta_r) = \{C : rank(C) \leq \lfloor |\mathcal{C}| \times \theta_r \rfloor\}$$

  where

  $$rank : \mathcal{C} \to \mathbb{N}$$

  is a function that ranks each $C \in \mathcal{C}$ according to their size. Suppose C_{max} is the largest cascade, i.e $\nexists C \in \mathcal{C} \setminus C_{max}$ such that $|C| > |C_{max}|$, then $rank(C_{max}) = 1$.

Finally, we state the formal definition of our problem as:

Problem 3. Given an observation and prediction times t_o and t_p, respectively, a set of observations of early events of cascades $\mathcal{O} = \{trace(C, t_o) : C \in \mathcal{C}\}$ and number d, we seek to learn a representation

$$\Phi : \mathcal{O} \to \mathbb{R}^d$$

$$subj.\ to$$

$$\min -\sum_{i=1}^{|\mathcal{O}|} y_i \log f(\Phi(O_i)) + (1 - y_i) \log(1 - f(\Phi(O_i)))$$

where $y_i \in \{1 = \texttt{viral}, 0 = \texttt{non-viral}\}$ is the label of the i^{th} observation $O_i \in \mathcal{O}$ computed at t_p and

$$f : \mathbb{R}^d \rightarrow \{0, 1\}$$

is a binary classifier that predicts the class of an observation of the early events $O_i = trace(C, t_o)$ of a cascade $C \in \mathcal{C}$ using the representation $\Phi(O_i)$.

6.3 The Learning Algorithm

The design of our algorithm is inspired by the observation that most viral cascades spread like a wildfire within the very first few hours. In contrast, non-viral cascades require several hours just to reach merely a handful of users. For instance, Fig. 6.2 shows the user coverage distribution of two hashtags in a 24-hour period, one viral (`#thingsigetalot`) and one not (`#bored`).

Some state-of-the-art studies [97, 89, 24] start from a similar assumption as in the above and develop elegant solutions based on *point processes*. Such techniques rely on the frequency (density) estimation of the rate of cascade growth during its observation period to predict its ultimate size after a certain period Δ.

Our approach is partially related, in the sense that it implicitly utilizes the rate of growth of the number of events within an observation period during the transformation of a cascade into a time series. However, it is completely network-agnostic. Based on our main premise, intuitively we seek to model the initial speed of a cascade (that is, the speed by which a cascade starts its spread) or the user reaction times at the early stage of the cascade, as well as its momentum. As we shall empirically demonstrate in Section 6.3.1, this is a strong signal for potential virality.

In Algorithm 2, we present the high-level steps required to train a CAS2VEC model that learns a representation of cascades for virality pre-

Algorithm 2: CAS2VECTRAIN $(\mathcal{C}_{train}, t_o, t_p, \theta, d, N_s, t_s)$

1 $\mathcal{L} = \emptyset$ /* Initialize a place holder for the label of cascades */
2 $\mathcal{T} = \emptyset$ /* Initialize a place holder for the set of observed cascades transformed into time series */
3 **for** $C_i \in \mathcal{C}_{train}$ **do**
4 $O_i = trace(C_i, t_o)$
5 $T_i = \text{TRANSFORMCASCADE}(O_i, N_s, t_s)$ /* The preprocessing step in Section 6.3.1 */
6 $size_i = |trace(C_i, t_p)|$
7 $L_i = \text{LABELCASCADE}(size_i, \theta)$ /* Label computed at t_p */
8 $\mathcal{T}.insert(T_i)$
9 $\mathcal{L}.insert(L_i)$
10 model = TRAINCNN$(\mathcal{T}, \mathcal{L}, d)$
11 return model

diction. Essentially the algorithm has the following major operations for each cascade C_i in our training data set $\mathcal{C}_{train}$:

- First, we extract the observation $trace(C_i, t_o)$ (line 4), where t_o is the observation time at which the observation period ends and the prediction starts;
- We discretize each observation $trace(C_i, t_o)$ by transforming it (line 5) into a format that can be fed to our classification task;
- We label (line 7) the cascade C_i as viral or not viral, based on the threshold θ according to the size of events ($size_i$ – line 6) observed at time $t_p = t_o + \Delta$, as discussed in the previous section.
- Finally, we train a model based on CNN that learns representation of cascades optimized for virality prediction (line 10) using the transformed cascades $\mathcal{T}$ and the associated labels $\mathcal{L}$.

During inference time, we use the model returned by Algorithm 2 to infer a representation of a given observation $O = trace(C, t_o)$ of a cascade $C \in$

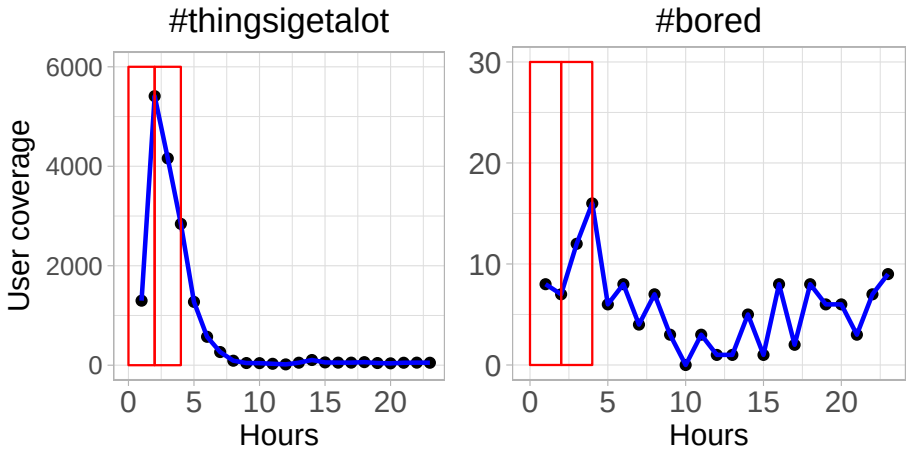


Figure 6.2: Two slices of size 2 hours, applied to the user coverage distribution of a viral hashtag (`#thingsigetalot`) and a non-viral one (`#bored`), which have reached 13711 and 43 users in an observation window size of 4 hours.

Algorithm 3: CAS2VECINFERENCE (`model`, O, N_s, t_s)

1 $T = \text{TRANSFORMCASCADE}(O, N_s, t_s)$
2 $\Phi(T) = \text{INFERCNN}(\texttt{model}, T)$ `/* Φ(T) ∈ ℝ^d is an embedding of T */`
3 $y = \text{PREDICTVIRALITY}(\Phi(T))$ `/* y is the predicted class 1 or 0. */`
4 return y

$\mathcal{C}_{test}$ and predict it, where $\mathcal{C}_{train} \cap \mathcal{C}_{test} = \{\}$. The overview of the inference procedure is shown in Algorithm 3. Its input is the trained CAS2VEC `model`, an observation O of a cascade C, the number of slices N_s and the slice size t_s. In line 1, we discretize the observed cascade O as required by the CAS2VEC input specification. Then, we infer the representation (line 2) of the transformed cascade T and finally the predicted state (line 3) of the cascade is returned. In the following we discuss the details of the aforementioned training and inference steps.

6.3.1 Pre-processing Cascades

Line 5 of Algorithm 2 takes an observation $O_i = trace(C_i, t_o)$ of a cascade C_i and two parameters known as number of slices N_s and slice size t_s. These

values are used to apply *slices* over O_i so as to discretize it. Therefore, as a result of applying slices the observation is divided into a collection of slices, i.e. equally-sized time windows according to N_s and t_s. For example, Fig. 6.2 illustrates an application of $N_s = 2$ slices having an equal size of $t_s = 2$ on top of an observation $O = trace(C, t_o = 4)$, visualized through red boxes. The size of the observation window t_o should be an integer multiple of t_s, such that the *number of slices* N_s is equal to t_o/t_s.

Based on the slices, we generate the following two kinds of pre-processed sequences:

Counter sequence the sequence of integers representing the number of events included in each slice:

$$C^{count} = [\ |trace(C, i \cdot t_s, (i+1) \cdot t_s)| : 0 \leq i < N_s\ ] \tag{6.4}$$

Constant sequence the sequence generated by discretizing every event within each slice to a constant value, *i.e.* by assigning each event within a slice the position of the slice itself.

$$C^{const} = [\ \lceil O(i)/t_s \rceil : 1 \leq i \leq |O| \wedge O = trace(C, t_o)\ ] \tag{6.5}$$

It can also be interpreted as a step function on the observed cascade $C^{const} = [\ step(O, i) = [i + 1 : i \cdot t_s \leq j \leq (i+1) \cdot t_s] : 0 \leq i \leq N_s\ ]$, as shown in Fig. 6.3.

For example, look again at Fig. 6.2 with cascades C_1 (`#thingsigetalot`) and C_2 (`#bored`). By considering an observation window size of 4 hours and a slice size of 2 hours, the counter sequences are equal to $C_1^{count} = [6\,709, 7\,002]$ and $C_2^{count} = [15, 28]$; in the former, there are 6 709 events in the first 2 hours, and 7 002 in the second 2 hours. In the later, the numbers

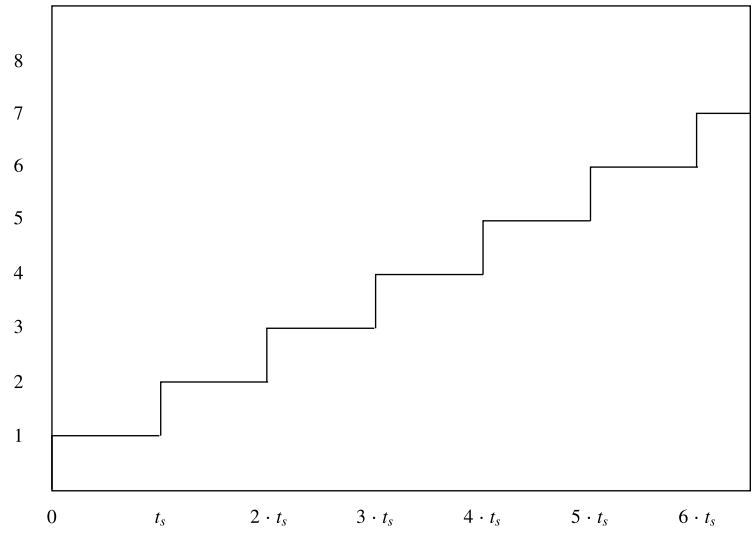


Figure 6.3: Constant sequence as a step function

are just 15 and 28. The constant sequences are equal to:

$$C_1^{const} = [\ \overbrace{[1,\ldots,1]}^{step1\ \ 6{,}709\ \ 1's}\ ,\ \overbrace{[2,\ldots,2]}^{step2\ \ 7{,}002\ \ 2's}\ ]$$

$$C_2^{const} = [\ \overbrace{[1,\ldots,1]}^{step1\ \ 15\ \ 1's}, \overbrace{[2,\ldots,2]}^{step2\ \ 28\ \ 2's}\ ]$$

Counter sequences and constant sequences have different predicting power. However, counter sequences are much faster to train as a result of a fixed length of training sequences, *i.e.* N_s, while constant sequences give us the flexibility of choosing larger values for the length of sequences at the expense of slower training time.

Based on our assumption regarding the dynamics of viral and non-viral cascades, we base our algorithm on the following conjecture:

Conjecture 6.3.1. Consider two cascades C_1 and C_2 and an absolute threshold θ. Given an observation t_o and a prediction window size Δ, if the cascade sizes of C_1 and C_2 at time $t_o + \Delta$ are such that $|C_1(t_o + \Delta)| \geq \theta$ and $|C_2(t_o + \Delta)| \ll \theta$, then $|C_1(t_o)| \gg |C_2(t_o)|$.

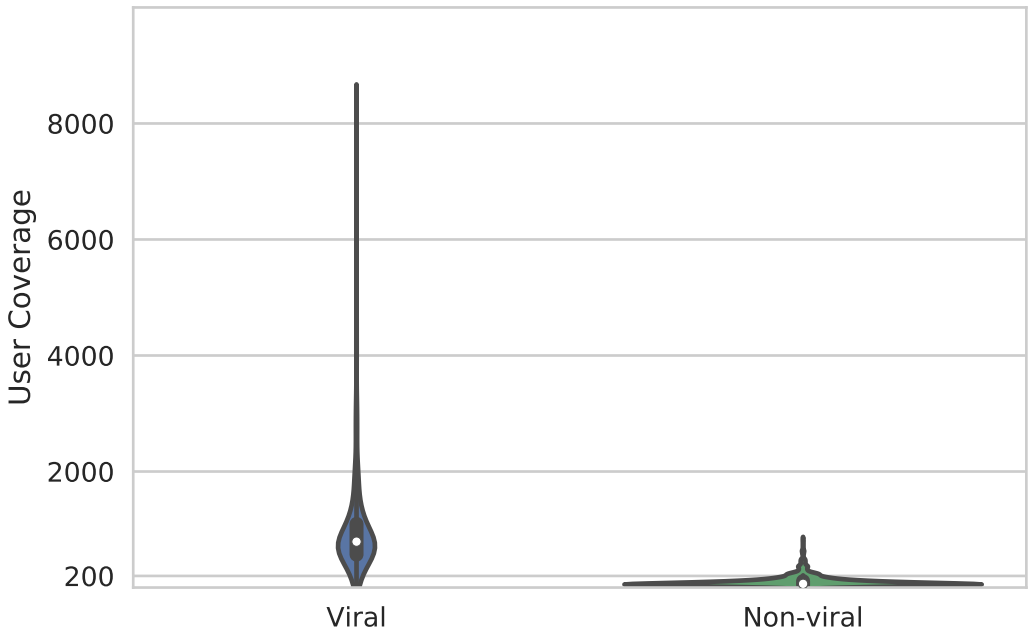


Figure 6.4: The distribution of the user coverage for the viral and non-viral classes. The user coverage distribution is computed at observation time t_o as $|C(t_o)|$ and virality is computed at prediction time $t_o + \Delta$. A cascade is viral if $|C(t_o + \Delta)| \geq 1,000$ and not-viral if $|C(t_o + \Delta)| < 1000$

According to the conjecture, within the observation window, we expect a significant number of events for viral cascades and very few of them for the non-viral ones. For example, looking again at Fig. 6.2, we have $13,711$ events for the viral hashtag `#thingsigetalot` and just 43 events for the non-viral hashtag `#bored` during the observation period. More generally, the user coverage distribution for the two classes, shown in Fig. 6.4, further establishes an empirical case for the conjecture.

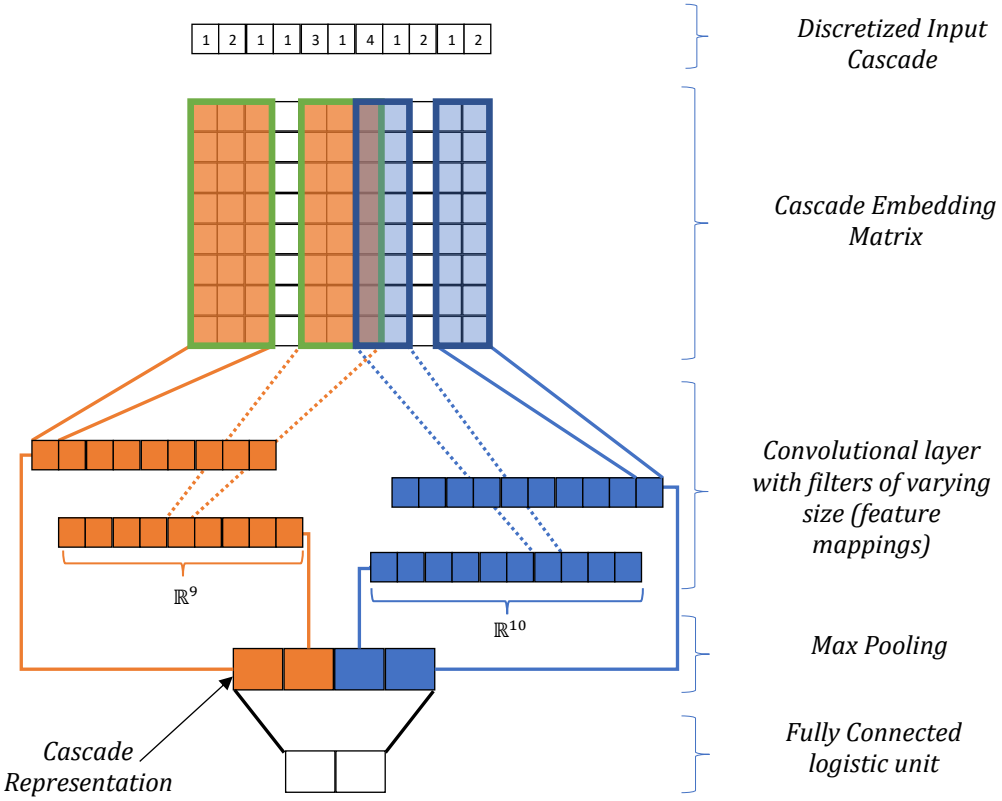


Figure 6.5: The CNN model adopted for cascade prediction

6.3.2 CNN model for cascade prediction

Once cascades are pre-processed using slices, we adopt the CNN model [42] to learn representation of cascades or features that are capable of predicting whether a cascade will go viral or not.

The architecture of the model that we adopted for cascade prediction is shown in Fig. 6.5 [42]. As discussed in Section 3.4, this model was originally proposed for sentence classification in natural language documents. Our choice of this model is inspired by recent studies that have shown the effectiveness of CNN for time-series classification tasks [96, 80], which have strong resemblance to fit our problem. In the following, we give a brief description of the model to show how it is translated to our problem.

Instead of words, the input to the model is a pre-processed cascade, the

"Discretized Input Sequence" part of Fig. 6.5. Next we have the "Cascade Embedding Matrix", which encodes each input by an embedding matrix. Next we have the "Convolutional layer" where we apply a set of filters followed by the "Max Pooling" layer. The same set of principles and operations are applied here as in Section 3.4.

During the training phase, we iteratively update the embedding matrix and the remaining model parameters ($\mathbf{h_i}, \mathbf{h}, b$ - Equations 3.18 and 3.20) until the binary cross-entropy loss function defined in Equation 3.19 is minimized. Once the optimal values are obtained, at inference time we fix the values of the model parameters to this values and compute a representation of the discretized input sequence at the max pooling layer.

Finally, it is the output of this layer that we consider as a *feature* or *representation* of an observed cascade that we use for predicting the virality of the the cascade at the final layer.

6.4 Experimental Evaluation

In this section, we report on the experiments we performed to evaluate our approach. Before discussing the actual results, we introduce the datasets that have been used as input; we discuss the competing approaches against which we compare our results; and finally, we describe the experiment settings.

6.4.1 Datasets

We have evaluated our approach over two well-known datasets:

- *Twitter*: This dataset has been commonly used for cascade prediction [97, 68]. It contains a full month of Twitter data from October, 7^{th} to November 7^{th}, 2011. There are a total of 166,076 tweets that have been retweeted at least 50 times.

- *Weibo*: This dataset contains 225,126 tweets recorded on the Chinese micro-blogging site Weibo [93, 94].

6.4.2 Baselines

We have compared our algorithm against three competing approaches; nevertheless, a well-known baseline [68] have not been included, because their source code is not available.

- SEISMIC: This is a recent, state-of-the-art study that predicts the popularity of tweets using a *self-exciting point process* model [97]. It estimates the infectiousness of a tweet at time t, based on the number of re-shares R_t at time t, then the estimated infectiousness is used to predict the ultimate size R_∞ of the tweet. We follow a similar strategy as [68] to label tweets based on R_∞, that is viral if and only if $R_\infty \geq \theta$. We have used the source code provided by the authors [4].

- Logistic Regression (LOR): This baseline has been used in previous studies [68, 12]. We use a set of features $X = [x(1), \ldots, x(N_s)]$ computed based on the notion of slices in Section 6.3.1, where $x(i)$ is the number of users in slice i and N_s is the number of slices.

- Linear Regression (LR): This is also a baseline similar to the one used in [97, 68]. It is specified as:

$$\log R_\infty = \log(\alpha \cdot R_{t_o}) + b + \epsilon,$$

  where ϵ is a noise term with Gaussian distribution. We apply a similar thresholding as we did with SEISMIC to label R_∞ as viral and non-viral, taking into account the log transformation.

[4] http://snap.stanford.edu/seismic/

6.4.3 Evaluation Settings

To evaluate the performance of our algorithm against the baselines, we have used the following settings. Recall that the prediction problem is based on an observation time t_o and a prediction window Δ. So, in all the reported results for all the classification algorithms, we have trained a single classifier for every given value of Δ. Furthermore, since the class distribution is highly skewed and the viral class is very rare, we use down-sampling in all the experiments.

During the training phase, we tune the hyper-parameters, e.g. the number and size of filters, using a *development set* (*dev-set*), $\mathcal{C}_{dev} \subset \mathcal{C}_{train}$, sampled from the training set $\mathcal{C}_{train}$. Once the hyper-parameters are tuned, we then fix the parameters at these values for all the experiments and evaluate the performance of the algorithms. Towards this end, we have used a 3-fold cross validation on an unseen test set $\mathcal{C}_{test}$ (which does not include the training and dev sets) and reported the average result along with the error margins.

Similar to [68], the evaluation metrics are F-score with $\beta = 3$ (since it is a rare class prediction), recall and coverage. In all the experiments, the threshold for labeling cascades is $\theta_a = 700$ that is equivalent to $\theta_r \approx 98\%$.

6.4.4 Virality Prediction

In the first set of experiments, we evaluate the performance of our algorithm and the baselines in predicting the virality of cascades based on a given observation t_o and prediction window Δ expressed in hours. Here, our goal is to evaluate the performance of algorithms in effectively classifying both classes as far as possible in the future. Fig. 6.6 reports the evaluation results. All the variants of our algorithm (CAS2VEC$_{\text{count}}$, CAS2VEC$_{\text{const}}$) outperform the baselines, and provide very similar results. The strongest

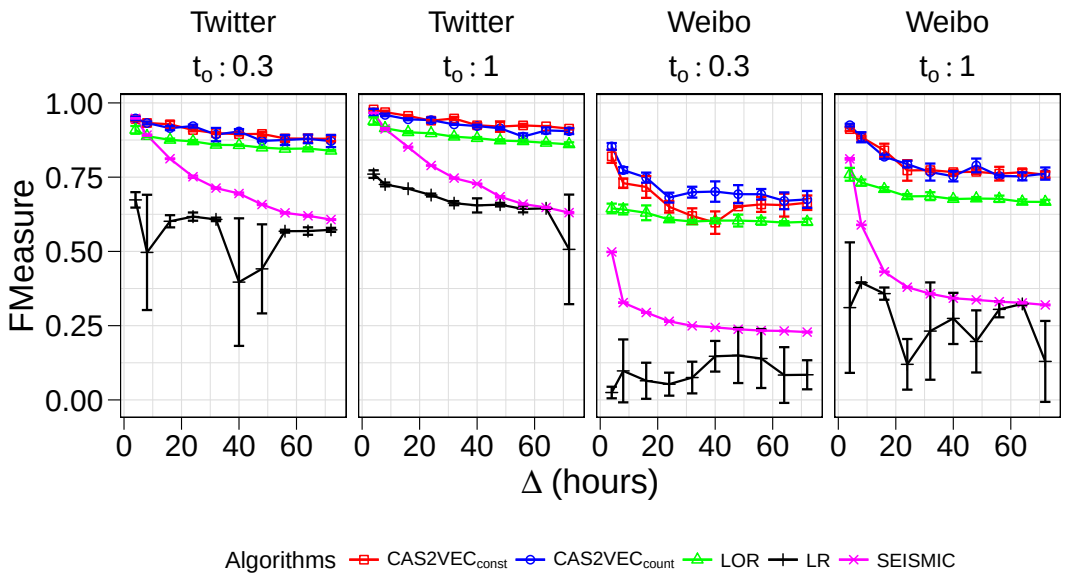


Figure 6.6: Virality prediction results for both of our datasets. For Twitter, *filter sizes* = 3, 5, 7 and for each filter we have 16 of them. For Weibo, *filter sizes* = 2, 4, 5, 7 and for each filter we have 64 of them. For both datasets, the size of the embedding matrix is 128, the number of units in the fully connected layer is 32, and the *number of slices* is 40.

baselines are SEISMIC and LOR; in the Twitter dataset, SEISMIC achieves F-scores between 94% and 60% for $t_o = 0.3$ hours and between 96% and 63% for $t_o = 1$ hour. LOR is more robust than SEISMIC and it achieves F-scores between 90% and 83% for $t_o = 0.3$ and between 93% and 86% for $t_o = 1$ hour. Whereas, CAS2VEC variants are much more robust in predicting far in the future than all the baselines and achieves F-scores between 97% and 88% and between 97% and 91% for $t_o = 0.3$ and $t_o = 1$ hours, respectively.

For the Weibo dataset, LOR achieves F-scores between 64% and 59% for $t_o = 0.3$ hour and between 75% and 66% for $t_o = 1$ hour. SEISMIC's performance on Weibo is poor and it achieves F-scores between 49% and 22% and between 81% and 31% for $t_o = 0.3$ and $t_o = 1$ hours, respectively. CAS2VEC, on the other hand, achieves a significantly higher performance, which is more than the performance of other baselines by at least 10%, *i.e.*

F-scores between 85% and 67% for $t_o = 0.3$ hour and between 92% and 76% for $t_o = 1$ hour.

In the following, unless stated otherwise, we focus on CAS2VEC$_{\text{count}}$, as it is faster to train.

The above experiments give us a perspective on how far an algorithm can effectively predict in the future stages of a cascade life. As we can see from the plots, performance decreases as Δ increases, as it is difficult to predict far in the future for both classes.

6.4.5 Early Prediction

The next step is to analyze how early in time virality can be predicted. Subbian et al. have observed that most of the events occur within twice the median virality time measured over all the cascades [68]. In our datasets, the median time to virality is 8 hours for Twitter and 17 hours for Weibo. Based on that, we select a distinct (but fixed) prediction time $t_p = t_o + \Delta$ ($t_o = t_p - \Delta$) for each of the dataset, i.e. $t_p = 16$ hours for Twitter and $t_p = 34$ hours for Weibo.

We then vary the size of the prediction window size Δ, from 1 hour to $t_p - 1$ hours to fix an observation time t_o and evaluate how early the algorithms can predict virality. In this setting, parameter Δ is similar to the time-to-virality parameter defined in [68]. Note that having fixed the prediction time, this means that the observation time t_o varies inversely w.r.t. the prediction windows size Δ, from $t_p - 1$ hours to 1 hour. In both cases, the variation step is 1 hour.

In the following experiment (Fig. 6.7), we evaluate the recall score only for the viral classes, that is measured by the fraction of viral cascades detected by an algorithm out of all the viral ones.

CAS2VEC obtains the best result for both datasets. As one might expect, all the algorithms achieve good results for larger values of t_o (small values

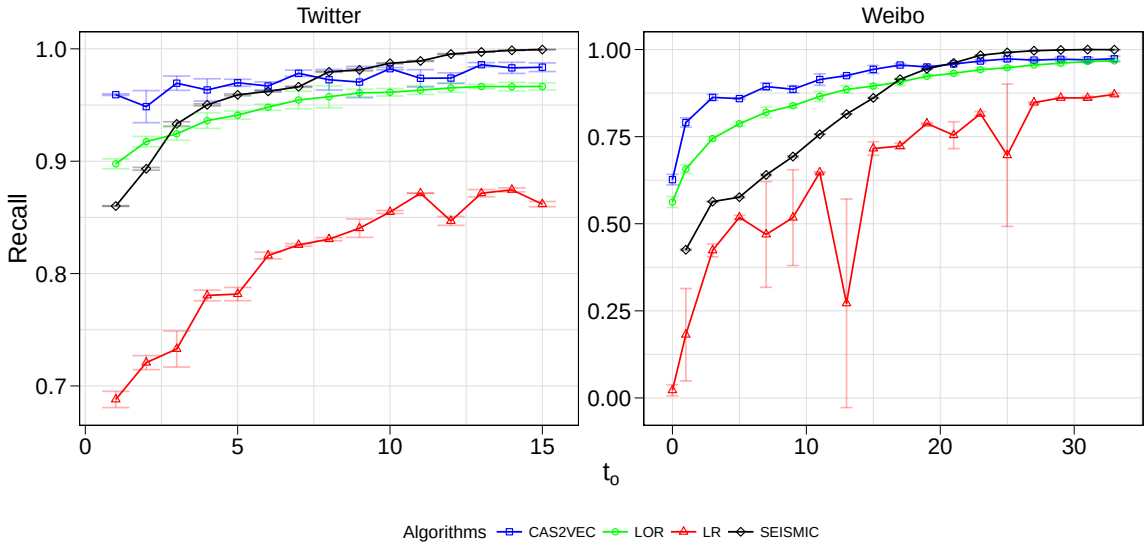


Figure 6.7: Evaluation results of early prediction experiments for the Twitter and Weibo datasets. The prediction time is fixed to 16 hours for Twitter and 34 hours for Weibo, and the same hyper-parameter values as Fig. 6.6 is used

of Δ). For example, all algorithms except linear regression achieve more than 96% recall in the Twitter dataset, with SEISMIC achieving the highest of all, i.e. 99%. Such result is trivial, however, and we want algorithms to be robust in their prediction as we increase Δ and the observation time t_o gets smaller.

As we approach $t_o \approx 0$ (only after a few activities have been detected), the performance of the baselines drop faster than CAS2VEC, which achieves the best recall. SEISMIC achieves the best results after $t_o = 7$ and $t_o = 25$, which is after observing for more than 7 and 25 hours for Twitter and Weibo respectively. However, for earlier observation points $t_o < 7$ (Twitter) and $t_o < 25$ (Weibo), SEISMIC achieve far worst ($\approx 10\%$ lower for Twitter and $\approx 20\%$ lower for Weibo) results than CAS2VEC; at $t_o = 1$, SEISMIC's recall is only 86% for Twitter and 42% for Weibo. At the same value $t_o = 1$, the other strong baseline, LOR, achieves 89% and 56% of recall, while CAS2VEC achieves 95% and 62% for Twitter and Weibo respectively.

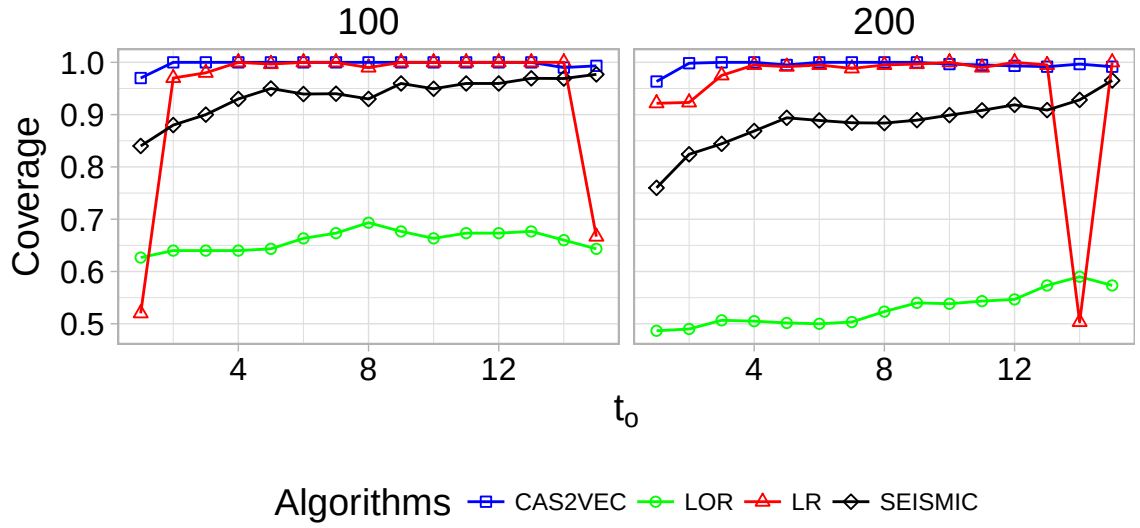


Figure 6.8: Break-out coverage for $k = 100$ and $k = 200$ for the Twitter dataset.

Besides the virality predictions shown previously, these experiments demonstrate that CAS2VEC is highly robust compared to the state-of-the-art method, SEISMIC, and the strong baseline, LOR, in predicting cascades virality as early as possible.

6.4.6 Break-out Coverage

One of the important tasks in cascade prediction is detecting break-out events. Towards this end, similar to [97, 68], we take the top-k viral cascades and evaluate the performance of algorithms in effectively covering such cascades in their prediction. That is, the fraction of correctly predicted cascades out of the top-k viral cascades.

The results of this experiment are reported in Fig. 6.8-6.9. Yet again, CAS2VEC consistently achieves a significant performance gain, specially as Δ increases or for smaller values of t_o (x–axis). Similar to the previous experiment, it is important to achieve a high coverage as $t_o \to 0$, and this is of vital importance in trend forecasting and rumor detection tasks. Note

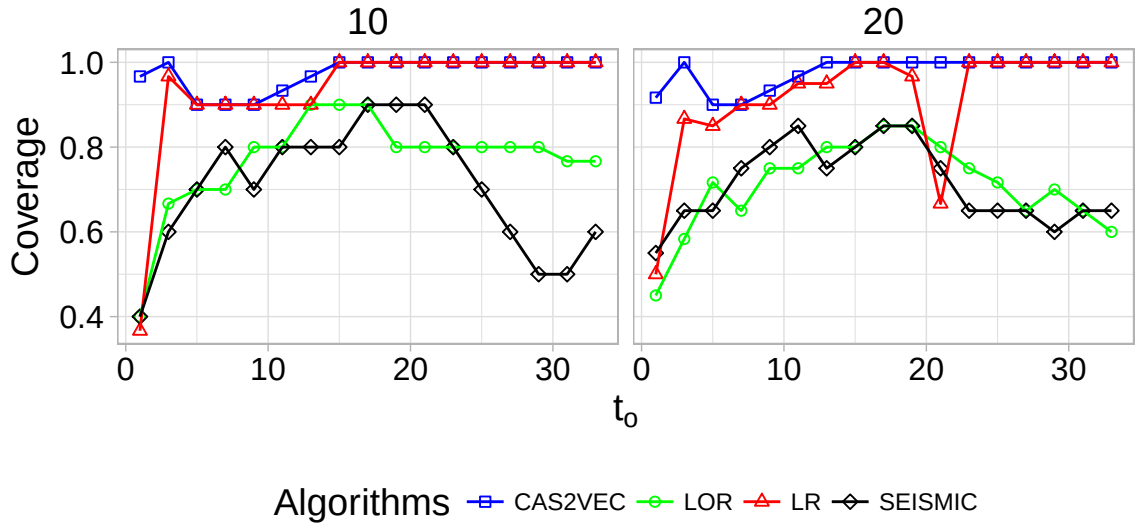


Figure 6.9: Break-out coverage for $k = 10$ and $k = 20$ for the Weibo dataset.

that even though LOR was a strong baseline in the earlier experiments, its performance degrades when it comes to detecting just the top-k break-out cascades. For Twitter, in particular at $t_o = 1$, the strongest baseline in this experiment achieves only 83% break-out coverage for $k = 100$, and 76% for $k = 200$. CAS2VEC, however, achieves a remarkable performance of 95% and 90% for $k = 100$ and $k = 200$, respectively. For the Weibo dataset, all the baselines score below 50% and 60%, whereas CAS2VEC achieves more than 90% for $k = 10$ and 20, respectively.

6.4.7 Effect of hyper-parameters

In order to further validate our proposal, we conducted two brief experiments on the effect of its hyper-parameters. First, we analyzed how the performance varies with the number of slices. As shown in Fig. 6.10, the performance increases as we increase the number of slices – up to a certain value. For Twitter, as we go from 10 to 30 the performance drops and starts to improve until we get to $N_s = 50$, which is the best spot; for

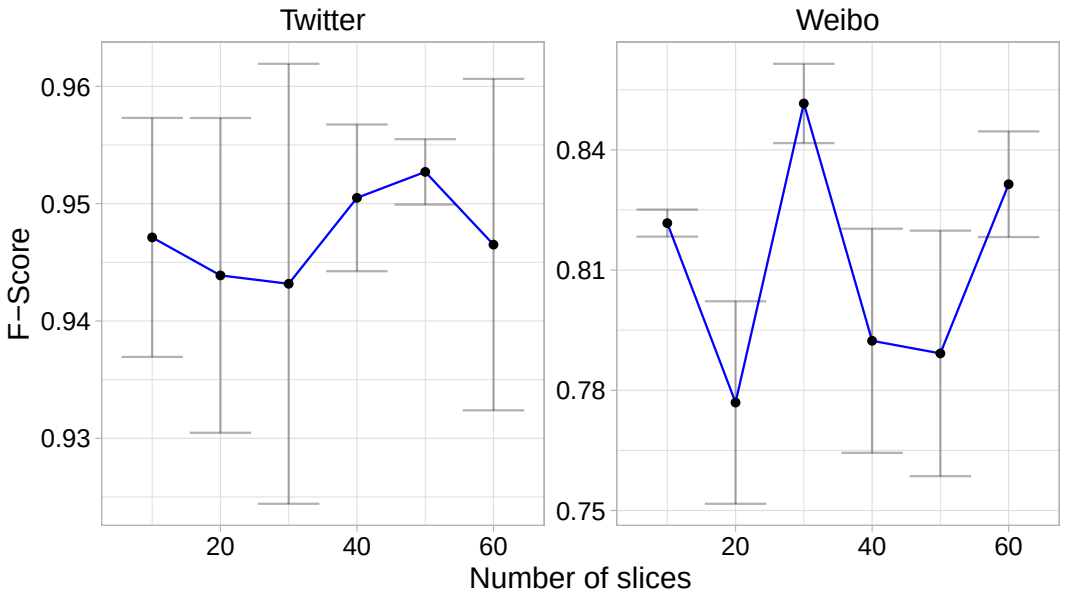


Figure 6.10: Effect of the number of slices on virality prediction at $t_o = 1$ hour and $\Delta = 12$ hours.

Weibo, the best F-score is achieved at $N_s = 30$. We have found out that values between 30 and 50 give the best results.

The other hyper-parameter of our algorithm is sequence length; in particular, it is the major factor in the run-time of our algorithm. Fig. 6.11 show the effect of sequence length (determined by t_o) for the two variants of our algorithms. Particularly CAS2VEC$_{\text{const}}$ requires more time to finish an epoch as we increase the sequence length. However, Fig. 6.12 shows that increasing the sequence length beyond a certain value (150 in the figure) does not give significant performance gain.

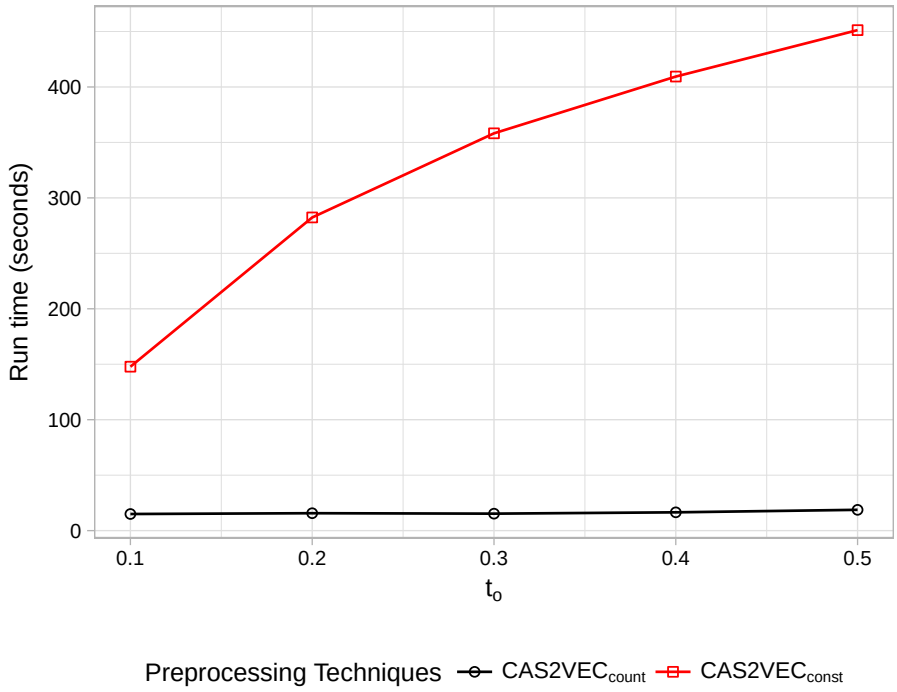


Figure 6.11: Effect of sequence length on running time.

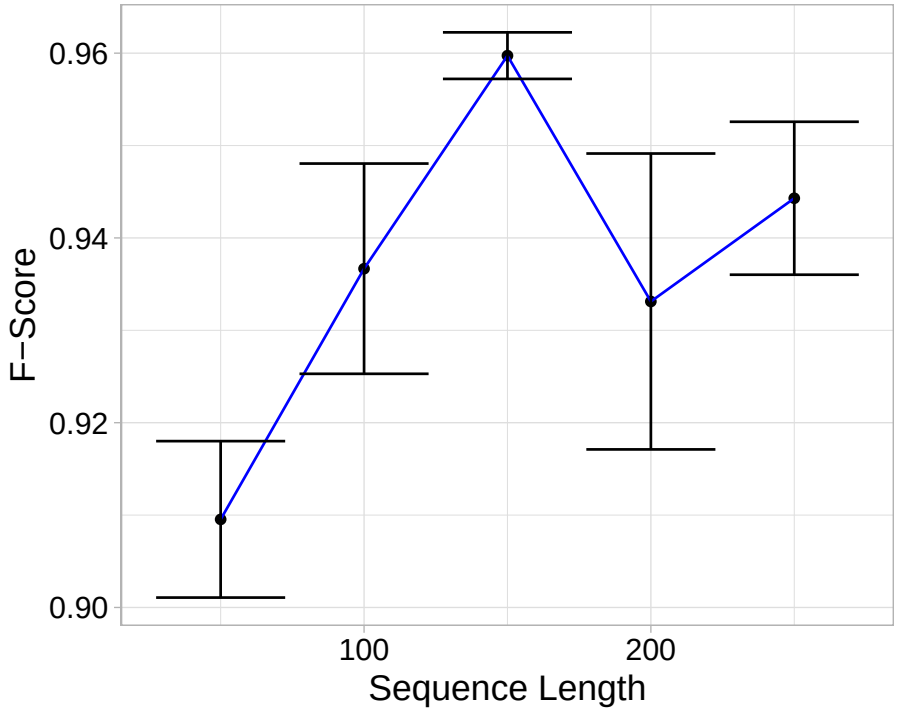


Figure 6.12: Effect of sequence length on virality prediction.

Chapter 7

State of the Art

In this thesis we have addressed the problem of representation learning in the context of *information networks*, focusing on two variants called:

1. Network Representation Learning
2. Cascade Representation Learning

In the former case we have shown the learned representations are general-purpose and can be applied to different kinds of problems that may arise in network analysis. In the latter case, on the other hand, the learning was carried out with virality prediction as a goal. This chapter covers selected state of the art studies in these two directions.

7.1 Network Representation Learning

As in many other areas, such as computer vision, natural language processing, the state-of-the-art in NRL is dominated by algorithms based on neural networks – neural NRL algorithms. Traditionally, dimensionality reductions based on matrix factorization (MF) techniques have ruled the domain. However, these techniques have two major limitations.

- they are linear models

- they do not scale

Information networks have a highly non-linear structure [77]; for this reason, exploiting non-linear learning models have proved to be much more effective than the linear MF methods. In addition, eigenvalue decomposition is usually at the core of these approaches, a technique that is very expensive for large graphs. Hence, MF is not the best choice for such graphs.

Since the seminal work of Perozzi et al.(DeepWalk [64] that has inspired several studies [28, 61, 66, 22, 11, 39, 17, 67, 90], we have witnessed an explosion of neural NRL algorithms that are scalable to graphs with millions of nodes and capable of capturing highly non-linear graph structures.

The basic idea of DeepWalk is inspired by the SkipGram algorithm from language model. As discussed in Chapter 3, the SkipGram is known for its effectiveness in learning low-dimensional latent representation of words that capture their distributional semantics. We have also pointed out that this model requires a linearly organized input and graphs are not one of them.

The goal of NRL is also to learn a low-dimensional representation of nodes that capture their context (local and global neighborhoods). In DeepWalk [64], the authors devise a walk sampling strategy to build a linear ordering of nodes that capture their context and compatible with the SkipGram input specification. During the sampling process, they uniformly draw a set $\mathcal{S}$ of truncated random walk sequences that effectively encode the local neighborhood information of nodes.

$$\mathcal{S} = \{\mathcal{S}_u : u \in V\}$$

$$\mathcal{S}_u = \{S_i : i = 1, \ldots, w\},$$

where $S_i = [s_1, \ldots, s_l]$ is a walk sequence, l is the maximum length of any walk sequence, and w is the number of walks to be sampled from each node $u \in V$.

Since random walks have a linear ordering of nodes, they are a simple yet novel trick to resolve graphs compatibility issue with SkipGram. Finally, akin to the DeepWalk model where a document corpus is consumed to create a word embedding, SkipGram consumes the walk corpus $\mathcal{S}$. The learning objective is then exactly the same as the one we specified in Section 3.3, which maximizes the log likelihood (Eq. 7.1) of seeing each context node $v_c \in \mathit{context}(S, v_t, s)$ of a target node v_t, where $S \in \mathcal{S}$:

$$\max \sum \log P(v_c|v_t) \tag{7.1}$$

Though DeepWalk is very effective and outperforms traditional techniques, it has severe limitations for weighted graphs, where uniform random walks are not the right choice. A follow up study by Grover et al. [28](Node2Vec) has identified this problem and suggested an extension of DeepWalk, which does biased random walks.

The biased random walks of Node2Vec are governed by two hyperparameters called p and q. The choice of this parameters will allow us to perform a biased random walk that could alternate or be balanced between breadth- or depth-first graph traversal strategies.

We have also proposed a technique, which we have covered in Section 4.3.1; a biased sequence sampling technique that draws inspiration from information diffusion processes. Regardless of the sequence sampling techniques, most techniques that are based on random walks have the same learning objective given in Eq. 7.1.

There are also techniques that are not based on random walks, and the pioneer is Line by Tang et al. [70]. The main goal of Line is to learn embeddings of nodes that preserve their first- and second-order proximity.

In preserving the first-order proximity, the objective is to make sure that the embedding of nodes that are connected by an edge are close to each other, in a manner similar to Laplacian eigenmaps [5]. More formally, given a weight (binary or real) matrix $\mathbf{W}$ of a graph, the objective is specified by the KL-divergence (Eq. 7.2) between an empirical distribution, $\tilde{p}(u_i, u_j)$ and a joint probability distribution $p(u_i, u_j)$ defined between u_i and u_j, where $(u_i, u_j) \in E$

$$- \sum_{(u_i,u_j)\in E} \tilde{p}(u_i, u_j) \log p(u_i, u_j) \tag{7.2}$$

$$\tilde{p}(u_i, u_j) = \frac{\mathbf{W}[\mathbf{i}, \mathbf{j}]}{\sum_{(u_i,u_k)\in E} \mathbf{W}[\mathbf{i}, \mathbf{k}]} \tag{7.3}$$

$$p(u_i, u_j) = \frac{1}{1 + \exp(\Phi_1(u_i)^T \cdot \Phi_1(u_j))} \tag{7.4}$$

where $\Phi_1(u)$ is the first-order embedding of node u.

In the second-order case, the intuition is that if two nodes have a similar set of common neighbors, their embeddings should be close to each other. Now, each node $u \in V$ has two roles, which are as itself and as a neighbor of another node $v \in V$; and hence $\Phi_2(u)$, and $\Phi_2'(u)$ denote its second-order embeddings as itself and as a neighbor, respectively. Then again, they specify the objective for the second order learning by KL-divergence as follows

$$- \sum_{(u_i,u_j)\in E} \tilde{p}(u_i, u_j) \log p(u_j|u_i) \tag{7.5}$$

where the $p(u_j|u_i)$ is the probability that node u_j is in the outgoing neighborhood of u_i and its specified by

$$p(u_j|u_i) = \frac{\exp(\Phi_2'(u_j)^T \cdot \Phi_2(u_i))}{\sum_{(u_i,u_k)\in E} \exp(\Phi_2'(u_k)^T \cdot \Phi_2(u_i))} \tag{7.6}$$

They use the negative sampling trick to make Eq. 7.6 tractable. At the end, they separately optimize the two objectives in Eq. 7.2 and 7.6 and concatenate the two embeddings, $\Phi_1(u)$ and $\Phi_2(v)$, as the output embedding $\Phi[\mathbf{u}]$ of each node $u \in V$.

A crucial limitation of the aforementioned techniques is that they can only capture a local view of nodes. The direct connections and a few hope neighborhood view, that is due to the truncated walk and diffusion simulations. One particular study [11](HARP) endeavored towards addressing this challenge by applying a hierarchical approach on top of the core component of the aforementioned algorithms. Instead of directly executing these algorithms on the input graph G, they first generate a hierarchical view of G by applying multiple levels of graph coarsening. Given the graph G and a threshold τ, they compute $\mathcal{G}$,

$$\mathcal{G} = [G_{=}(V, E), G_1 = (V_1, E_1), \ldots, G_l = (V_l, E_l)]$$

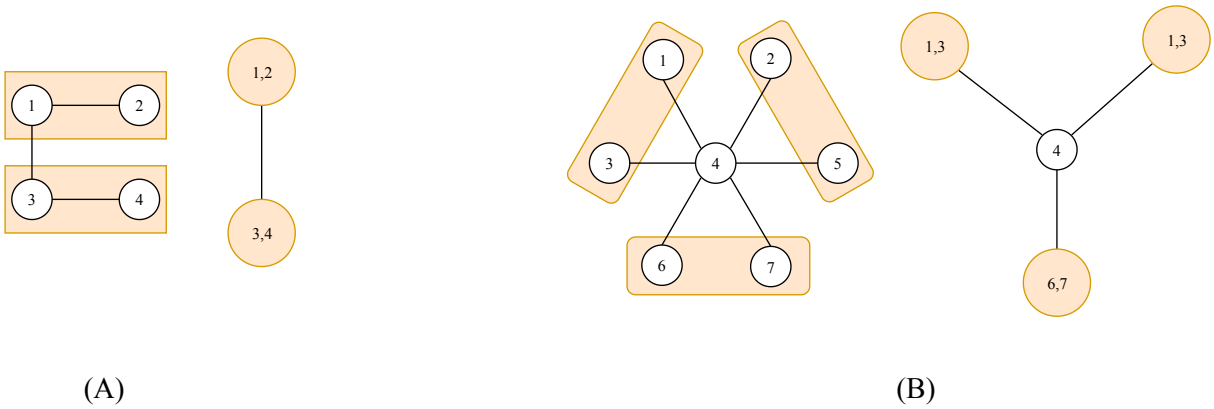
where each graph G_i with $i > 1$ is obtained by applying graph a coarsening function on the previous graph G_{i-1}, until the number of vertexes $|V_l|$ is smaller than a threshold τ. The first graph G_0 is the original one

Learning is performed in the reverse direction starting from the last graph $G_l = (V_l, E_l)$, by executing the core component of any algorithm h, say $h =$ DEEPWALK, as $\mathbf{R}_l = h(G_l, \emptyset) \in \mathbb{R}^{|V_l| \times d}$, where

$$h : \mathcal{G} \times \mathbb{R}^{n_{i+1} \times d} \rightarrow \mathbb{R}^{n_i \times d}$$

is trained on a finer level $G_i = (V_i, E_i) \in \mathcal{G}$, where $n_i = |V_i|$ by starting from the representations $\mathbf{R}_{i+1} \in \mathbb{R}^{n_{i+1} \times d}$ learned on the coarser graph $G_{i+1} = (V_{i+1}, E_{i+1}) \in \mathcal{G}$, where $n_{i+1} = |V_{i+1}|$.

Since G_l is the coarsest graph, its embedding, $\mathbf{R}_l = h(G_l, \emptyset)$, is learned from a random initialization. For the rest, learned representations $\mathbf{R}_{i+1}$ are propagated to $\mathbf{R}_i$, *i.e.*, $\mathbf{R}_i = h(G_i, \mathbf{R}_{i+1})$ from the coarse G_{i+1} to the fine G_i graph until we get to the finest or original graph G, *i.e.*, $\Phi = \mathbf{R} = h(G, \mathbf{R}_0)$.



(A) (B)

Figure 7.1: HARP's Graph coarsening techniques (A) - edge coarsening and (B) - star coarsening

For the graph coarsening, they proposed two techniques, known as *edge* and *star coarsening* shown in Fig. 7.1. The edge coarsening collapses two incident nodes u and v of an edge $(u, v) \in E' \subseteq E$ as one super node uv, such that in a given step no node u is collapsed more than once.

Besides the locality, another problem with LINE is that the first- and second-order objectives are trivially combined. In a follow up study, Wang et al. have proposed a more principled way for jointly optimizing both objectives [77]. Their approach, called SDNE, is a semi-supervised model using a deep autoencoder architecture, as shown in Fig. 7.2. The gist of the joint objective of the model is to minimize the loss function in Eq. 7.7.

$$\mathcal{L} = \alpha \cdot \mathtt{tr}(\Phi^T \mathbf{L} \Phi) + ||(\mathbf{W} - \tilde{\mathbf{W}}) \otimes \mathbf{B}||_F^2 + \gamma \sum_l ||\mathbf{H}_{enc}^l||_F^2 + ||\mathbf{H}_{dec}^l||_F^2, \quad (7.7)$$

where $\mathbf{H}^l$ is part of the model parameters (weight matrix of the l^{th} hidden layer) and $\mathbf{L}$ is the graph Laplacian matrix.

The first term of the equation used for preserving first-order proximity is a supervised loss specified using Laplacian eigenmaps. The second term is a reconstruction loss for preserving second-order proximity, and $\mathbf{B}$ is introduced to avoid the trivial and unwanted solution obtained by recon-

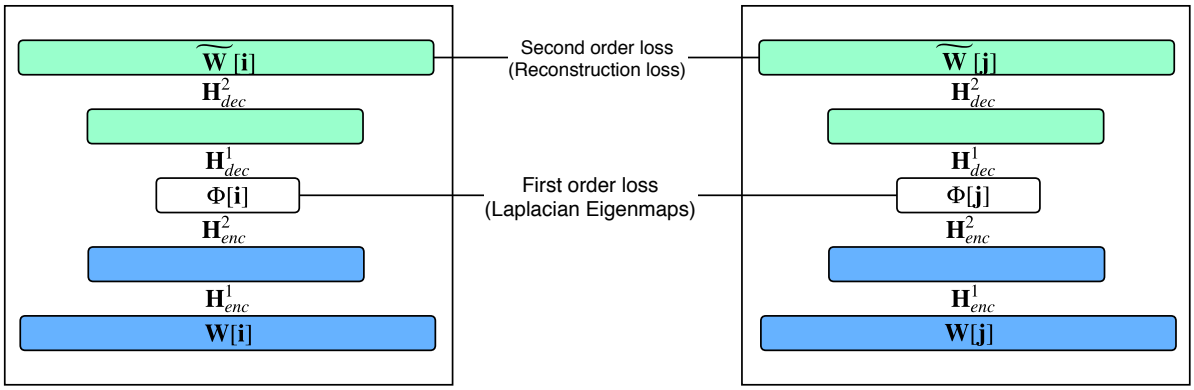


Figure 7.2: SDNE model

structing the zeros of $\mathbf{W}$, which is what we have already covered in Section 3.3. Finally, an L_2 regularization term is added to avoid over-fitting. Note that weights are shared between the left and right autoencoders. The model has two hyper-parameters α and γ, which control the contribution from the first order and regularization loss terms.

All the techniques that we have covered so far consume just the topology of a given graph. However, most real-world graphs have high-quality side information associated with nodes and edges. In the following, we cover results achieved by techniques capable to use auxiliary information to the network topology. TriDnr [61] is one of the first examples to couple topological, attribute and label information for NRL. In addition to the set of nodes V and edges E, a graph G formulation in TriDnr consists of a document corpus

$$\mathcal{D} = \{D_u : u \in V\}$$

$$D_u = \{w_i : 1 \leq i \leq |D_u|, w_i \in \mathcal{V}\}$$

and a partial set of labels $\mathcal{C} = L \cup U$ associated with nodes of the graph, where L and U denote the set of labeled and unlabeled nodes, respectively.

Then, the model objective in Eq. 7.8 is specified with two necessary and a third optional term.

$$\mathcal{L} = (1 - \alpha) \cdot \text{DeepWalk} + \alpha \cdot \text{Doc2Vec} + \alpha \cdot h \tag{7.8}$$

The first term has exactly the same formulation as DeepWalk (more accurately – a weighted DeepWalk), and the second one introduces a loss term sensitive to preserving textual information associated to nodes, similar to the Doc2Vec [47] model for sentence representation learning. On the other hand, the objective of the weighted Doc2Vec formulation in TriDnr is to maximize the log-likelihood of a particular word $w_i \in D_u$ given u.

$$\text{Doc2Vec} = \sum \log P(w_i|u) \tag{7.9}$$

Finally, the optional term h is used for semi-supervised learning using the labels of each node $u \in L$. Here, the goal is yet again to maximize the log-likelihood of a word $w_i \in D_u$ given the class label c_u of each labeled node u.

$$h = \sum_{u \in L} P(w_i|c_u) \tag{7.10}$$

Both Eq. 7.9 and 7.10 are computed using the standard softmax form similar to Eq. 7.6. Note that, if $L = \emptyset$, the model becomes purely unsupervised. Furthermore, it has a hyper-parameter α that governs the contribution of each term.

Provided its effectiveness as opposed to purely structural techniques, one of TriDnr's limitation is that when $\mathcal{D}$ is very sparse its gain is marginal. To tackle this problem, in one of our papers [66] we propose a technique called gat2vec that samples more textual information via truncated random walks by modeling $\mathcal{D}$ as a separate bipartite graph $G_{att} = (V_{att}, E_{att})$. That is, for every word $w \in D_u$, we create a node $u_w \in V_{att}$ and we build an undirected edge $(u, u_w) \in E_{att}$. Then, we run walk samplings both on top of G and G_{att} and obtain the walk sequences $\mathcal{S}$ and $\mathcal{S}_{att}$, respectively.

Finally, the two walk sequences are combined into a single walk corpus $\mathcal{S}' = \mathcal{S} \cup \mathcal{S}_{att}$ and we optimize the standard DeepWalk objective. We have shown that this simple trick is very effective and obtains a significant performance gain over TriDnr.

Like the purely structural variants, different and complementary non-random walk techniques have been proposed for attributed graph embeddings. sne [55] proposes a technique using the standard deep feed-forward neural network architecture. Their novelty is that the network is fed information coming from topology and attributes (features). The input to their algorithm is a one hot-encoded vector of nodes and a generic feature vector that encodes different kinds of side information. The two inputs are projected into a structural $\Phi_{\mathbf{s}}[\mathbf{u}]$ and $\Phi_{\mathbf{f}}[\mathbf{u}]$ feature embedding of each node u, which is then jointly passed to a feed-forward network. Finally, the output of the model is a conditional probability distribution $p(v|u)$ over each node $v \in V$ given a node $u \in V$. $p(\cdot|u)$ is a distribution predicting the probability that each node $v \in V$ has an edge with u. Ultimately, the objective is to train the feed-forward network to maximize the log-likelihood of the graph $\mathcal{L}(G;\Theta)$ that is similar to what we have seen in several of the earlier models.

The problem with this approach is that even though it uses side information, the optimization is constrained on the specific objective of predicting links or G. This makes it suitable for link prediction but not for other network analysis tasks. Most recent achievements have addressed this issue and are able to optimize a more general objective.

One study [23](dane) extended the deep autoencoder model of sdne to explicitly exploit both the structural and side (attributes) information. Intuitively, their objective function optimizes the loss function in Eq. 7.7 for structure and attributes. More concretely, there are two weight matrices $\mathbf{W}$ and $\mathbf{F}$ corresponding to two structural proximity and feature matrixes,

respectively. Therefore, a structural component of the model works on optimizing Eq. 7.7 using $\mathbf{W}$ and the attribute component using $\mathbf{F}$, and this will enable them to obtain the embeddings $\Phi_{\mathbf{s}}[\mathbf{u}]$ and $\Phi_{\mathbf{f}}[\mathbf{u}]$ corresponding to the two components, respectively. However, $\Phi_{\mathbf{s}}[\mathbf{u}]$ and $\Phi_{\mathbf{f}}[\mathbf{u}]$ live into different spaces and the trivial solution of concatenating them is suboptimal [23]. For this reason, they incorporate an extra loss term that ensures the two embeddings are consistent.

A complementary study(ANRL) that was published in the same venue also uses a different autoencoder like architecture for the same purpose [95]. Similar to DANE they have the two ground-truth matrices, $\mathbf{W}$ and $\mathbf{F}$, but $\mathbf{F}$ could be an adjacency or some higher order proximity matrix. However, unlike DANE instead of feeding two inputs to two separate autoencoders in ANRL they only have a single encoder that takes nodes features $\mathbf{f_i}$ from $\mathbf{F}$ as an input. Then, they replace the decoder of the standard autoencoder with two branches that tries to predict $\tilde{\mathbf{W}} \approx \mathbf{W}$ and $\tilde{\mathbf{F}} \approx \mathbf{F}$. Essentially, the intuitive objective of their algorithm is to minimize the prediction errors on $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{F}}$ in a manner similar to Eq. 7.7.

Most of the techniques we have seen so far are obviously unsupervised in relation to node labels. Even if this is desirable to extract the inherent patterns in the data and could be useful across multiple problems, sometimes one might be interested in task-specific embeddings of nodes or can also improve the quality of the embeddings by incorporating labels whenever possible.

An collection of works have responded to this need and proposed different kinds of semi-supervised techniques that utilize partial label information [90, 54, 44, 29, 91, 15].

A method called PLANETOID formulates the representation learning as a two way learning problem [90]. An input is denoted by $\mathbf{F} = [\mathbf{f_1}, \ldots, \mathbf{f_n}]$, the feature vector of nodes. Suppose $\phi_p(\ldots \mathbf{x} \ldots)$ denotes a feed-forward

neural network (multilayer layer perceptron - MLP) with p layers of linear transformations and non-linear activations. Then, the current input feature $\mathbf{f_u}$ of node u is fed into two MLPs:

$$\mathbf{H}[\mathbf{u}] = \phi_p(\ldots \mathbf{f_u} \ldots)$$

$$\Phi[\mathbf{u}] = \phi_q(\ldots \mathbf{f_u} \ldots)$$

$\Phi[\mathbf{u}] \in \mathbb{R}^{n \times d}$ itself is again fed to

$$P(v|\Phi[\mathbf{u}])$$

$$\mathbf{H}'[\mathbf{u}] = \phi_t(\ldots \Phi[\mathbf{u}] \ldots),$$

where $P(v|\Phi[\mathbf{u}])$ is the standard softmax classifier used to learn the graph context nodes v of the target node u in an unsupervised manner similar to DeepWalk. Finally, $\mathbf{h'_u} = \mathbf{H}'[\mathbf{u}]$ and $\mathbf{h_u} = \mathbf{H}[\mathbf{u}]$ are combined together and used to predict the class label of node u:

$$P(c_u|(\mathbf{h'_u} \oplus \mathbf{h_u}))$$

The two classifiers $P(v|\Phi[\mathbf{u}])$ and $P(c_u|(\mathbf{h'_u} \oplus \mathbf{h_u}))$ are intended to predict a context node $v \in \{Context(S, u, s) : S \in \mathcal{S}\}$ and class label c_u of node u, where $\mathcal{S}$ is obtained by sampling random walks from the graph G like DeepWalk and Node2Vec. The model is trained by a back-propagation algorithm based on the misclassification errors on the unsupervised (graph context) and supervised (node label) predictors.

Recently, a new kind of architecture called *graph convolutional networks*(GCN) have received a considerable attention for supervised and semi-supervised network representation learning. Normally, convolutional neural networks (CNN) have been successfully and widely used on inputs like images, texts, signals, and videos. Nonetheless, due to the requirement of a regular grid like inputs, there was no straightforward way of adopting them to irregular structures like graphs. This has triggered recent

studies [15, 44, 29, 76, 91] in an attempt to generalize CNN to any kind of shape. They are generally categorized into spectral and non-spectral approaches [75].

The main challenge in designing graph convolutional neural networks is how we define localized features that are "space"- and "translation"-invariant, similar to standard CNNs. In spectral approaches, these issues are addressed by answering what convolutions are at the low-level, which corresponds to a diagonal operator in the eigenspace of the graph Laplacian [48], which is the Fourier space. The diagonal operator $\mathbf{g}_\theta = \texttt{diag}(\theta)$ that is used as a filter is parameterized by $\theta \in \mathbb{R}^n$. The application of a graph convolution on an input feature $\mathbf{f_u} \in \mathbb{R}^n$ using the above filter is given by

$$\mathbf{g}_\theta \star \mathbf{f_u} = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{f_u} \tag{7.11}$$

where $\mathbf{U}$ is the matrix associated with the eigenvectors of the normalized graph Laplacian matrix

$$\mathbf{L} = \mathbf{I_N} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\Lambda\mathbf{U}^T \tag{7.12}$$

and $\mathbf{U}^T\mathbf{f_u}$ is the Fourier transform of the input $\mathbf{f_u}$ [15, 76, 44, 48].

However, computing the eigenvectors of $\mathbf{L}$ is not computationally possible for large graphs. Noting this limitation, in a subsequent study [15] the graph filter $\mathbf{g}_\theta$ of Eq. 7.11 is approximated by K–order Chebyshev polynomials $T_K(\cdot)$ as:

$$\mathbf{g}_{\theta^c} \star \mathbf{f_u} = \sum_{k=1}^{K} \theta_k^c T_k(\tilde{\mathbf{L}})\mathbf{f_u} \tag{7.13}$$

where

$$\tilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}_N,$$

λ_{max} is the largest eigenvalue of $\tilde{\mathbf{L}}$ and the parameter θ_k^c is the coefficient of the k^{th}–order Chebyshev polynomial. It has been shown that stacking

a number h of graph convolutions based on the aforementioned filter for a given target node u successfully convolves over the h–hop neighborhood of u. Again, the formulation in Eq. 7.13 has been further simplified by simply considering a first order Chebyshev polynomials, that is $K = 1$ for scalability and yet manages to achieve state-of-the-art results [44].

Other studies related to GCN follow a non-spectral approach and propose algorithms that are impressively scalable, having been successfully applied on graphs with billions of nodes and edges [29, 91]. Given a node u and a number k associated to the execution of the k^{th} layer, to estimate the current embedding $\Phi^{\mathbf{k}}[\mathbf{u}]$ of u, first they aggregate the previous embeddings $\Phi^{\mathbf{k-1}}[\mathbf{v}]$ of the neighbor nodes v of node u, where $v \in N(u) = in(u) \cup out(u)$ into a single u's neighborhood representation $\Psi^{\mathbf{k-1}}[\mathbf{u}]$:

$$\Psi^{\mathbf{k-1}}[\mathbf{u}] = \text{AGG}(\{\Phi^{\mathbf{k-1}}[\mathbf{v}] : v \in N(u)\}) \tag{7.14}$$

and AGG is any kind of trainable aggregation function over a set of unordered vectors. Three set of aggregation functions, which are mean, LSTM, and pooling are proposed in these studies [29, 91]. Second, $\Psi^{\mathbf{k-1}}[\mathbf{u}]$ is concatenated with the previous embedding $\Phi^{\mathbf{k-1}}[\mathbf{u}]$ of the node u itself as

$$\Phi'^{\mathbf{k-1}}[\mathbf{u}] = \Psi^{\mathbf{k-1}}[\mathbf{u}] \oplus \Phi^{\mathbf{k-1}}[\mathbf{u}] \tag{7.15}$$

Finally, $\Phi^{\mathbf{k}}[\mathbf{u}]$ is obtained by feeding $\Phi'^{\mathbf{k-1}}[\mathbf{u}]$ into a feed-forward neural network as

$$\Phi^{\mathbf{k}}[\mathbf{u}] = \sigma(\mathbf{H}^{\mathbf{k}}\Phi'^{\mathbf{k-1}}[\mathbf{u}] + b) \tag{7.16}$$

where σ is a non-linear activation function. After the final pass (layer) $k = K$, the normalized vector

$$\Phi[\mathbf{u}] = \frac{\Phi^{\mathbf{k}}[\mathbf{u}]}{||\Phi^{\mathbf{k}}[\mathbf{u}]||_2} \tag{7.17}$$

is returned as the embedding of node u.

As directly working on $N(u)$ is not convenient with respect to the time complexity and memory footprint of such algorithm [29, 91] due to a varying size of $N(u)$, Hamilton et al. proposed to sample a fixed number of neighbors in a uniform way [29]. It has been later proposed, however, to sample a fixed number of neighbors according to a score function $score(N(u))$, which measures each neighbor node's $v \in N(u)$ influence on node u [91].

Similar to the spectral approaches, an interesting aspect of these methods is that at the k^{th} layer of a target node u, the k–hope neighborhood information is used to compute $\Phi^{\mathbf{k}}[\mathbf{u}]$. In this way, k–hope node features are effectively propagated. Finally, these GCN architectures can be trained in a semi-supervise fashion based on a set of target node labels.

In the non-spectral GCN approaches the fixed number of neighborhood samples considered for scalability will force them to ignore the complete neighborhood space. To gracefully deal with this limitation, the notion of *graph attention networks*(GAT) has been introduced in [75].

The GAT technique has a resemblance with GCN techniques, in the sense that an estimation of an embedding $\Phi[\mathbf{u}]$ of a target node u involves gathering features $\mathbf{f_v}$ of neighboring nodes $v \in N(u)$ through self attention mechanism. The core component of GAT is this attention mechanism, which enables them to consider an entire neighborhood of a node as a opposed to the fixed number of sampled neighbors considered in [29, 91].

Like most of the supervised and semi-supervised models that we have seen, the input of GAT is specified by a feature matrix $\mathbf{F} \in \mathbb{R}^{n \times r}$. Then, a MLP with L layers and self-attentions are successively applied on $\mathbf{F}$ to obtain an embedding of nodes $\Phi \in \mathbb{R}^{n \times d}$, $d \ll r$. We use $\mathbf{F^l}$ to denote the output of the l^{th} layer of the MLP, when applied to $\mathbf{F}$. Following our notational convention $\mathbf{f_v^l}$ or $\mathbf{F^l}[\mathbf{v}]$ corresponds to the v^{th} row of $\mathbf{F^l}$. In a given layer l, a weight matrix $\mathbf{H}_l \in \mathbb{R}^{r_{l-1} \times r_l}$ shared by all nodes is used

to apply a linear transformation (Eq. 7.18) of its input $\mathbf{f_u^{l-1}} \in \mathbb{R}^{r_{l-1}}$ (the output of the previous layer) and for the base case, or at the first layer $l = 1$, the input is $\mathbf{F}[\mathbf{u}]$.

$$\mathbf{x_u^l} = \mathbf{H^l f_u^{l-1}} \tag{7.18}$$

After the linear transformation, each node u attends to all of its neighboring nodes $v \in N'(u)$, where $N'(u) = N(u) \cup \{u\}$, to obtain a normalized attention coefficient $\alpha_{uv} \in \mathbb{R}$ as

$$e_{uv} = \texttt{att}(\mathbf{x_u^l}, \mathbf{x_v^l}) \tag{7.19}$$

$$\alpha_{uv} = \frac{\exp(e_{uv})}{\sum_{w \in N'(u)} \exp(e_{uw})} \tag{7.20}$$

Where the attention *att* layer parameterized by a weight vector $\mathbf{h^l} \in \mathbb{R}^{2r_l}$ is defined using a `LeakyReLU` as

$$\texttt{att}(\mathbf{x_u^l}, \mathbf{x_v^l}) = \texttt{LeakyReLU}(\mathbf{h^{l^T}} \cdot (\mathbf{x_u^l} \oplus \mathbf{x_v^l})) \tag{7.21}$$

Finally, the attention weighted linear combination of $\mathbf{x_v^l}$ of each neighbor node $v \in N'(u)$ is passed through non-linear function as follows to obtain $\mathbf{f_u^l}$

$$\mathbf{f_u^l} = \sigma(\sum_{v \in N'(u)} \alpha_{uv} \mathbf{x_v^l}) \tag{7.22}$$

where σ is a non-linear activation function. This model is extended by using *multi-head* attention mechanism to ensure that the learning process is stable [75]. A multi-head attention technique simply employs K independent self-attention mechanisms, which yields α_{ij}^k, $k = 1, \ldots, K$ attention coefficients that are used in concatenated/aggregated form to obtain $\mathbf{f_u^l}$. Eventually, for an L-layer GAT model, the value $\mathbf{f_u^L}$ is used as an embedding $\Phi[\mathbf{u}]$ of node u and it is trained using the labels of nodes.

7.2 Cascade Representation Learning

Traditionally, cascade representation learning (CRL) amounts to building hand-crafted sets of features for a cascade. Normally, the features are extracted from the underlying graph and cascades. The graph features include different kinds of structural features of the early starters of a cascade. Such features include for example in- and out-degree, community affiliation, structural roles and so on [82, 83, 12, 53]. Besides, features associated with the cascade itself has also been used to represent cascades. These include content features, time features, original poster features [12] and so on. Ultimately, the combination of such representations are used for the sole purpose of predicting cascades, regardless of the formulation, which is classification or regression. Before discussing CRL methods in Section 7.2.2 that aim at automatically extracting representation of cascades, in the following section we first give a brief overview on the cascade prediction task itself to provide a richer context.

7.2.1 Overview on Cascade Prediction

In general, the problem of cascade prediction can be considered from either a macroscopic or a microscopic perspective [86].

In the macroscopic case, usually the goal is to predict the final state of the entire cascade. Research efforts in this direction have formulated the prediction task as a regression – predicting the potential size a cascade will ultimately grow to [97, 73, 92, 53, 69], or as a classification task – predicting whether a cascade will become popular (viral) or not [82, 83, 12, 68, 13, 36]. As we have discussed already, most of the methods in both cases have been based on either topological information of the early starters and/or on features manually crafted from the cascades.

In the microscopic case, given the current state of the cascade, we are

interested in predicting what will happen next, *i.e.* who will be infected next? at what time? [81, 35, 86]. This approach is particularly relevant in applications like product recommendation [86]. For example, given a set of users that have purchased a product, who is the user that is highly likely to make the same purchase?

Even though most of the early studies have relied on manually extracted features, recent efforts leveraged the power of neural networks to automatically learn features for both the macroscopic and microscopic settings. The methods that are discussed in the following section embed cascades and/or users in a latent continuous vector space and use such embeddings for the prediction task at hand. Some of these works automatically learn an intrinsic single representation that captures the cascade pattern [35, 53, 37], while others learn a representation of the early starters during the observation period of the cascade and use different strategies, which will be covered below, to aggregate those into a single representation of a cascade [78, 86, 9].

7.2.2 Methods

The apparent problem with techniques based on manual feature engineering is that usually they are dependent on prior (expert) knowledge and external factors to identify the highly predictive features. The recent success of deep learning techniques in different fields, however, inspired several studies to leverage their power to automatically extract representation of cascades for predicting their future state. In this survey of the state-of-the-art we consider representative techniques exploiting neural networks for cascade representation learning.

The algorithm DeepCas in [53] introduced a *recurrent neural network* (RNN), or more specifically a *gated recurrent neural network* (GRU), to learn a representation that can predict the ultimate size of a cascade. A

graph $G_t = (V_t, E_t)$, where $V_t \subseteq V$ and $E_t \subseteq E$, is used to model an observation $trace(C, t) = [(u_1, t_1), \ldots, (u_i, t_i = t)]$ of a cascade C at time t with a sequence of users $[u_1, \ldots, u_i]$. Then, multiple random walks are sampled from G_t in a manner similar to DeepWalk and Node2Vec. Once sampled, the set of walks are fed into a recurrent neural network with attention to learn a single representation of the cascade under consideration. The learned representation is then used to predict the size of the cascade $|trace(C, t')|$ at the prediction point $t' = t + \Delta$. The model is trained by minimizing the mean square error (MSE) between the predicted size y_c of the cascade and the true size $|trace(C, t')|$.

In a similar line as DeepCas, another method called TopoLSTM by Wang et. al. [78] proposed to use another family of RNN's called LSTM. Besides using a different kind of RNN, in this work they have extended the LSTM architecture to TopoLSTM that is capable of encoding a *directed acyclic graph* (DAG). The DAG's are associated to a cascade and are intended to capture the structure of the cascade in G.

Given a graph G and an observation $trace(C, t) = [(u_1, t_1), \ldots, (u_i, t_i = t)]$ of a cascade C at time t, to learn a representation of the cascade they utilize an induced directed acyclic subgraph $G_t = (V_t, E_t)$ of G. Where $V_t = \{u_1, \ldots, u_i\} \subseteq V$ and $E_t \subseteq E_{t-1} \cup E$ contains edges from previous time step $t - 1$ and new edges $(u, v) \in E$ that were added to G_t in either of the following two conditions:

- If $u \in V_{t-1}$ succeeded to spread a contagion to $v \in V \setminus V_{t-1}$.
- If u made an attempt to spread a contagion to v.

In the next step, they combine the embedding of nodes in the TopoLSTM architecture in a way that captures the structure of the cascade in relation to G. This allows them to project the cascade into a single representation that is used to predict the user that is likely to be infected at

time $t+1$. Besides the representation of the cascade, the TopoLSTM model is parameterized by two kinds of user embeddings associated to a users state, which are 'active' and 'inactive'. Thus their model not only learns an embedding of cascade but also the users. Ultimately, TopoLSTM is trained based on the prediction error on the ground truth user u_{i+1}, $C(t_{i+1}) = (u_{i+1}, t_{i+1} > t)$ of each training cascade C.

A more recent study has incorporated a structure attention model with LSTM parameterized by both node and cascade embeddings [81]. The LSTM is used to learn the sequential pattern of the cascade, and the attention mechanism is used for capturing node structural information. They have a similar problem setting as TopoLSTM where they want to predict the node that is likely to be infected at time $t_{i+1} > t$, based on the observation $trace(C,t) = [(u_1,t_1),\ldots,(u_i,t_i = t)]$ of the cascade at t. The structural attention $\mathbf{a_v}$, computed over the inactive neighbors of a node v infected at $t_j \leq t$, $C(j) = (u_j = v, t_j)$, is combined with its current representation $\Phi[\mathbf{v}]$ and the previous state $\mathbf{h_c^{j-1}}$ of the cascade through the gating mechanism of LSTM's to obtain the current state

$$\mathbf{h_c^j} = \texttt{gate}(\mathbf{a_v}, \mathbf{h_c^{j-1}}, \Phi[\mathbf{v}]).$$

Finally, the current state $\mathbf{h_c^j}$ is used to predict the user u_{j+1} that is likely to be infected at next time step t_{j+1} in C.

The final state $\mathbf{h_c^i}$ can then be considered as a representation of the observed cascade that summarizes the information encoded in the cascade at an observation time $t_i = t$.

This representation $\mathbf{h_i^c}$ is then used to predict the next state u_{i+1} of the cascade C after the observation period t_i, or at t_{i+1}. As they have a similar objective as the previous two methods, their algorithm is trained in a similar way.

One of the strong assumptions in the above methods is that they con-

sider all infected users to be equally active and equally likely to spread a contagion. Yang et al. have relaxed this assumption and devise an attention mechanism to extract active users and consider these users to have a better chance of spreading the contagion [86]. Thus, the infection of a new user u is not equally attributed to all the previously infected users, but to the subset of them that are active. These set of active users denoted by $act(u)$ are potentially responsible for infecting u. The core idea behind their algorithm lies in identifying the active users in a cascade C at a certain time t_i in the life of C and using their embedding to predict the user to be infected at t_{i+1}. To obtain the active embedding $\Phi_{act}[\mathbf{u_i}]$ of a user u_i, $C(i) = (u_i, t_i = t)$, they first compute a normalized attention coefficient α_{ij} for all the previously infected users u_j in $\{C(j) = (u_j, t_j) : t_j < t\}$, similar to the attention mechanism we have seen in Eq. 7.20. Then, the attention weighted sum of the embeddings $\Phi_{act}[\mathbf{u_j}]$ of the previously infected users u_j is taken as $\Phi_{act}[\mathbf{u_i}]$:

$$\Phi_{act}[\mathbf{u_i}] = \sum_{\{C(j)=(u_j,t_j):t_j<t\}} \alpha_{ij}\Phi_{act}[\mathbf{u_j}]$$

Finally, the embedding $\Phi_{act}[\mathbf{u_j}]$ of the set of active users $u_j \in act(u_i)$ is combined with $\Phi_{act}[\mathbf{u_i}]$ using CNN to predict the next user u_{i+1} in C. Like the previous microscopic methods, their model is trained based on the softmax classification error on the ground truth user u_{i+1}, $C(t_{i+1}) = C(u_{i+1}, t_{i+1})$.

Almost all of the previous methods rely on the presence of the network structure, for example in [81] building the DAG associated with a cascade requires knowledge of the underlying graph. As we have been repeatedly argued throughout the thesis, it is possible to simply make use of the information available in the training cascades and avoid any assumption regarding the knowledge of the underlying graph [35]. Moreover, their objective is not only to predict the node u_{i+1} in the next step, but also the

actual next infection time t_{i+1}. Similar to some of the above methods, they employ LSTM's along with point processes for representing the cascades and ultimately predicting the tuple (u_{i+1}, t_{i+1}).

Chapter 8

Conclusions

In this thesis we have addressed the problem of representation learning, focusing on networks and cascades, in the context of information networks. We have proposed novel algorithms with the relaxed assumption that the network structure under consideration may be partially or completely unknown. The assumption is inspired by real-world scenarios where access to information networks is limited or completely unavailable.

For instance, followers and friendship links of social networks can only be partially accessed due to privacy policies. In addition, because of quotas, access to the crawler API's of the social networks is restricted to a few calls per unit of time. Therefore, one has to wait several weeks or even months before obtaining the partial network structure for a few thousand users. These issues are a challenge for business providing services based on social network data.

For this reason, we have developed several algorithms that are resilient to the lack of the above information; we present three of them in this thesis. Particularly, we present two novel algorithms for network representation learning and one for cascade representation learning.

In general, the goal of network representation learning is to project nodes into a latent continuous vector space that is dense and preserves the

original neighborhood information. However, when the network topology is partially or completely hidden, the notion of neighborhood becomes illusive. Hence our main challenge is how to account for such neighborhood of nodes in such a precarious situation. Our contribution is a collection of novel techniques that tackle this problem and approximate or estimate the neighborhood information of nodes.

To achieve this, our techniques took inspiration from previous results that empirically show the correlation between the properties of the network and users activities performed on top of it. In particular, we consider user activities that are recorded as diffusion events (cascades), such as shares, retweets and hashtags.

In the first network representation learning algorithm, called MINERAL, we follow the assumption that the network structure might be partially accessible. Thus we simply utilize the cascades as a complementary information along with the provided neighborhood information. That is, we simply sample artificial cascades from the partially observed network and combine them with the observed cascades to learn the representation of nodes. Following one of the aforementioned findings of previous studies, MINERAL is designed based on the SKIPGRAM model, which considers that two nodes belong to the same context (neighborhood) if they tend to closely co-occur within the sampled and observed cascades.

In the second algorithm called NETTENSOR, instead, we assume that the network structure is completely hidden and we only have access to nodes activity. For this reason, we first propose several methods that estimate nodes neighborhood or proximity and also extract different kinds of features merely based on the only available information, cascades. We finally exploit the estimated neighborhood information and the extracted features to jointly learn representation of nodes.

We have performed several experiments and evaluated the performance

of our NRL methods by comparing them against strong methods that make use of the complete graph. In some cases, such as MINERAL, we have achieved better performance than the state-of-the-art; in other cases, such as in NETTENSOR, we have achieved comparable results in spite of the missing information.

The last but not the least contribution is a novel cascade representation learning algorithm (CRL) called CAS2VEC. In CRL, the main goal is essentially to find embeddings of cascades that are useful for predicting their future state. CAS2VEC is a CRL method for cascade virality (popularity) prediction. Prior to CAS2VEC, most techniques have relied on manually crafted features taken from the cascade and network structure, and hence requires knowledge about the such network. Our method is network-agnostic and can automatically extract features using convolutional neural networks that are highly effective for virality prediction. We have carried out several experiments and compared CAS2VEC against state-of-the-art methods and other widely-used strong baselines. CAS2VEC consistently outperforms them in all kinds of the experiments we have carried out.

Acknowledgment

First and foremost, I'm eternally thankful for my God and Saviour, God the Father, God the Son, God the Holy Spirit for the beautiful gift of life and all the undeserved extra blessings.

Second, I would like to express my sincere gratitude for my advisor Prof. Alberto Montresor for his continuous support. This work would not have been successful if it was not for his knowledge, kindness, motivation and patience during my study. In addition, I would like to express my sincere gratitude for Prof. Sarunas Girdzijauskas who was my co-supervisor during a couple of visits at KTH Royal Institute of Technology.

I would like to thank my colleagues Nasrullah Sheikh, Cristian Consonni, Leila Bahri, and Amira Soliman for the invaluable knowledge and experience that I have gained from you all during our meetings and collaborations.

I am thankful for my beloved family, *Aba* and *Mami*, my sisters *Meazi* (plus her husband *Engedaye*) and *Lalaye* (plus her husband *Wondesho*), and my brother *Abi* who walked with me through continuous prayers and emotional support.

Thank you to all my friends in Ethiopia, Italy, and Sweden with whom I have shared a number of unforgettable moments.

Bibliography

[1] Nesreen K. Ahmed, Ryan Rossi, John Boaz Lee, Xiangnan Kong, Theodore L. Willke, Rong Zhou, and Hoda Eldardiry. Learning role-based graph embeddings. *CoRR*, abs/1802.02896, 2018.

[2] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Cascade-based community detection. In *Proc. of the 6th ACM Int. Conf. on Web Search and Data Mining*, WSDM '13, pages 33–42. ACM, 2013.

[3] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247. Association for Computational Linguistics, 2014.

[4] Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. Matrix reordering methods for table and network visualization. *Computer Graphics Forum*, 35(3):693–716, 2016.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 585–591, Cambridge, MA, USA, 2001. MIT Press.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[7] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[8] Simon Bourigault, Cedric Lagnier, Sylvain Lamprier, Ludovic Denoyer, and Patrick Gallinari. Learning social network embeddings for predicting information diffusion. In *Proc. of the 7th ACM Int. Conf. on Web Search and Data Mining*, WSDM '14, pages 393–402. ACM, 2014.

[9] Simon Bourigault, Sylvain Lamprier, and Patrick Gallinari. Representation learning for information diffusion through social networks: An embedded cascade model. In *Proc. of the Ninth ACM Int. Conf. on Web Search and Data Mining*, WSDM '16, pages 573–582. ACM, 2016.

[10] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. iSAX 2.0: Indexing and mining one billion time series. In *Proc. of ICDM'10*, pages 58–67, Washington, DC, USA, 2010. IEEE Computer Society.

[11] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: hierarchical representation learning for networks. *CoRR*, abs/1706.07845, 2017.

[12] Justin Cheng, Lada Adamic, P. Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *Proc. of WWW'14*, pages 925–936, New York, NY, USA, 2014. ACM.

[13] Peng Cui, Shifei Jin, Linyun Yu, Fei Wang, Wenwu Zhu, and Shiqiang Yang. Cascading outbreak prediction in networks: a data-driven approach. In *KDD*, 2013.

[14] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

[16] Inderjit S. Dhillon and Suvrit Sra. Generalized nonnegative matrix approximations with bregman divergences. In *Proc. of the 18th Int. Conf. on Neural Information Processing Systems*, NIPS'05, pages 283–290. MIT Press, 2005.

[17] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 135–144, New York, NY, USA, 2017. ACM.

[18] Nan Du, Le Song, Ming Yuan, and Alex J. Smola. Learning networks of heterogeneous influence. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2780–2788. Curran Associates, Inc., 2012.

[19] Daniel R. Figueiredo, Leonardo Filipe Rodrigues Ribeiro, and Pedro H. P. Saverese. struc2vec: Learning node representations from structural identity. *CoRR*, abs/1704.03165, 2017.

[20] J. R. Firth. The technique of semantics. In *Transactions of the Philological Society*, pages 36–73, 1935.

[21] J. R. Firth. A synopsis of linguistic theory 19301955. In *In Studies in linguistic analysis*, pages 1–32, 1957.

[22] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proc. of the 2017 ACM Conf. on Information and Knowledge Management*, CIKM '17, pages 1797–1806. ACM, 2017.

[23] Hongchang Gao and Heng Huang. Deep attributed network embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3364–3370. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[24] Shuai Gao, Jun Ma, and Zhumin Chen. Modeling and predicting retweeting dynamics on microblogging platforms. In *Proc. of WSDM'15*, pages 107–116, New York, NY, USA, 2015. ACM.

[25] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. *CoRR*, abs/1105.0697, 2011.

[26] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Trans. Knowl. Discov. Data*, 5(4):21:1–21:37, February 2012.

[27] Manuel Gomez Rodriguez, Jure Leskovec, and Bernhard Schölkopf. Structure and dynamics of information pathways in online media. In *Proc. of the 6th ACM Int. Conf. on Web Search and Data Mining*, WSDM '13, pages 23–32. ACM, 2013.

[28] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proc. of the 22Nd ACM Int. Conf. on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864. ACM, 2016.

[29] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[30] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[31] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[32] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: Structural role extraction & mining in large graphs. In *Proc. of the 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '12, pages 1231–1239. ACM, 2012.

[33] M.E. Hochstenbach. A jacobi davidson type method for the generalized singular value problem. *Linear Algebra and its Applications*, 431(3):471 – 487, 2009. Special Issue in honor of Henk van der Vorst.

[34] Xiao Huang, Jundong Li, and Xia Hu. Label informed attributed network embedding. In *Proc. of the Tenth ACM Int. Conf. on Web Search and Data Mining*, WSDM '17, pages 731–739. ACM, 2017.

[35] Mohammad Raihanul Islam, Sathappan Muthiah, Bijaya Adhikari, B. Aditya Prakash, and Naren Ramakrishnan. DeepDiffuse: Predicting the 'who' and 'when' in cascades. In *IEEE International Conference on Data Mining*, ICDM'18, pages 1055–1060, 2018.

[36] Maximilian Jenders, Gjergji Kasneci, and Felix Naumann. Analyzing and predicting viral tweets. In *WWW*, 2013.

[37] Zekarias T. Kefato, Nasrullah Sheikh, Leila Bahri, Amira Soliman, Alberto Montresor, and Sarunas Girdzijauskas. CAS2VEC: network-agnostic cascade prediction in online social networks. In *Fifth International Conference on Social Networks Analysis, Management and Security, SNAMS 2018, Valencia, Spain, October 15-18, 2018*, pages 72–79. IEEE, 2018.

[38] Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. Deepinfer: Diffusion network inference through representation learning. In *Proc. of the 13th Int. Workshop on Mining and Learning With Graphs*, MLG'17. ACM, August 2017.

[39] Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. Mineral: Multi-modal network representation learning. In *Proc. of the 3rd International Conference on Machine Learning, Optimization and Big Data*, MOD'17. ACM, September 2017.

[40] Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. REFINE: Representation learning from diffusion events. In *Proc. of the 4th Conference on Machine Learning, Optimization and Data science*, LOD'18. Springer, September 2018.

[41] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proc. of the Ninth ACM Int. Conf. on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146. ACM, 2003.

[42] Yoon Kim. Convolutional neural networks for sentence classification. In *Proc. of EMNLP'14*, pages 1746–1751, 2014.

[43] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[44] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[45] Yi-An Lai, Chin-Chi Hsu, Wen Hao Chen, Mi-Yen Yeh, and Shou-De Lin. Prune: Preserving proximity and global ranking for network embedding. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5257–5266. Curran Associates, Inc., 2017.

[46] Sylvain Lamprier, Simon Bourigault, and Patrick Gallinari. Extracting diffusion channels from real-world social data: A delay-agnostic learning of transmission probabilities. In *Proc. of the 2015 IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining 2015*, ASONAM '15, pages 178–185. ACM, 2015.

[47] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[48] Yann LeCun. Graph embeddings, content understanding,and self-supervised learning. 2018.

[49] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401:788–791, 1999.

[50] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Proc. of the 13th Int. Conf. on Neural In-*

formation Processing Systems, NIPS'00, pages 535–541. MIT Press, 2000.

[51] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proc. of the 15th ACM Int. Conf. on Knowledge Discovery and Data Mining*, KDD '09, pages 497–506. ACM, 2009.

[52] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proc. of the 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '09, pages 497–506. ACM, 2009.

[53] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. DeepCas: An end-to-end predictor of information cascades. In *Proc. of WWW'17*. Int. World Wide Web Conferences Steering Committee, 2017.

[54] Jiongqian Liang, Peter Jacobs, and Srinivasan Parthasarathy. SEANO: semi-supervised embedding in attributed networks with outliers. *CoRR*, abs/1703.08100, 2017.

[55] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *CoRR*, abs/1705.04969, 2017.

[56] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[57] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of the 26th Int. Conf. on Neural Information Processing Systems*, NIPS'13, pages 3111–3119. Curran Associates Inc., 2013.

[58] Seth A. Myers and Jure Leskovec. On the convexity of latent social network inference. In *Proc. of the 23rd Int. Conf. on Neural Information Processing Systems - Volume 2*, NIPS'10, pages 1741–1749. Curran Associates Inc., 2010.

[59] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proc. of the 22Nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '16, pages 1105–1114. ACM, 2016.

[60] Pentti Paatero and Unto Tapper. Positive matrix factorization: A nonnegative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[61] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1895–1901. AAAI Press, 2016.

[62] Ashwin Paranjape, Robert West, Leila Zia, and Jure Leskovec. Improving website hyperlink structure using server logs. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pages 615–624, New York, NY, USA, 2016. ACM.

[63] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2014, pages 1532–1543, 2014.

[64] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proc. of the 20th ACM Int. Conf. on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710. ACM, 2014.

[65] Eldar Sadikov, Montserrat Medina, Jure Leskovec, and Hector Garcia-Molina. Correcting for missing data in information cascades. In *Proc. of WSDM'11*, pages 55–64. ACM, 2011.

[66] Nasrullah Sheikh, Zekarias T. Kefato, and Alberto Montresor. GAT2VEC: representation learning for attributed graphs. *Computing*, April 2018.

[67] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and Philip S. Yu. Heterogeneous information network embedding for recommendation. *CoRR*, abs/1711.10730, 2017.

[68] Karthik Subbian, B. Aditya Prakash, and Lada Adamic. Detecting large reshare cascades in social networks. In *Proc. of WWW'17*, pages 597–605. Int. World Wide Web Conferences Steering Committee, 2017.

[69] Gabor Szabo and Bernardo A. Huberman. Predicting the popularity of online content. *Commun. ACM*, 53(8):80–88, August 2010.

[70] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. *CoRR*, abs/1503.03578, 2015.

[71] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proc. of the 15th ACM Int. Conf. on Knowledge Discovery and Data Mining*, KDD '09, pages 817–826. ACM, 2009.

[72] P. Thruesen, J. echk, B. Sezec, R. Castalio, and N. Kanhabua. To link or not to link: Ranking hyperlinks in wikipedia using collective attention. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1709–1718, Dec 2016.

[73] Oren Tsur and Ari Rappoport. What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities. In *WSDM*, 2012.

[74] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 2008.

[75] Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[76] Priyesh Vijayan, Yash Chandak, Mitesh M. Khapra, and Balaraman Ravindran. Fusion graph convolutional networks. *CoRR*, abs/1805.12528, 2018.

[77] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proc. of the 22Nd ACM Int. Conf. on Knowledge Discovery and Data Mining*, KDD '16, pages 1225–1234. ACM, 2016.

[78] Jia Wang, Vincent W. Zheng, Zemin Liu, and Kevin Chen-Chuan Chang. Topological recurrent neural network for diffusion prediction. In *ICDM*, pages 475–484, 2017.

[79] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. Link prediction in social networks: the state-of-the-art. *Sci. China Inform. Sci.*, 58(1):1–38, 2015.

[80] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. *CoRR*, abs/1611.06455, 2016.

[81] Zhitao Wang, Chengyao Chen, and Wenjie LI. A sequential neural information diffusion model with structure attention. In *Proceedings of*

the 27th ACM International Conference on Information and Knowledge Management, CIKM '18, pages 1795–1798, New York, NY, USA, 2018. ACM.

[82] L. Weng, F. Menczer, and Y.-Y. Ahn. Virality prediction and community structure in social networks. *Sci. Rep.*, 3(2522), 2013.

[83] Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Predicting successful memes using network and community structure. In Eytan Adar, Paul Resnick, Munmun De Choudhury, Bernie Hogan, and Alice H. Oh, editors, *ICWSM*. The AAAI Press, 2014.

[84] Robert West, Ashwin Paranjape, and Jure Leskovec. Mining missing hyperlinks from human navigation traces: A case study of wikipedia. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1242–1252, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[85] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network representation learning with rich text information. In *Proc. of the 24th Int. Conf. on Artificial Intelligence*, IJCAI'15, pages 2111–2117. AAAI Press, 2015.

[86] Cheng Yang, Maosong Sun, Haoran Liu, Shiyi Han, Zhiyuan Liu, and Huanbo Luan. Neural diffusion model for microscopic cascade prediction. *CoRR*, abs/1812.08933, 2018.

[87] Dejian Yang, Senzhang Wang, Chaozhuo Li, Xiaoming Zhang, and Zhoujun Li. From properties to links: Deep network embedding on incomplete graphs. In *Proc. of the 2017 ACM on Conf. on Information and Knowledge Management*, CIKM '17, pages 367–376. ACM, 2017.

[88] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *2012 IEEE 12th International Conference on Data Mining*, pages 745–754, Dec 2012.

[89] Shuang-Hong Yang and Hongyuan Zha. Mixture of mutually exciting processes for viral diffusion. In *Proc. of ICML'13*, pages II–1–II–9. JMLR.org, 2013.

[90] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016.

[91] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973, 2018.

[92] Linyun Yu, Peng Cui, Fei Wang, Chaoming Song, and Shiqiang Yang. From micro to macro: Uncovering and predicting information cascading process with behavioral dynamics. *2015 IEEE International Conference on Data Mining*, pages 559–568, 2015.

[93] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. Social influence locality for modeling retweeting behaviors. In *Proc. of IJCAI'13*, pages 2761–2767. AAAI Press, 2013.

[94] Jing Zhang, Jie Tang, Juanzi Li, Yang Liu, and Chunxiao Xing. Who influenced you? predicting retweet via social influence locality. *ACM Trans. Knowl. Discov. Data*, 9(3):25:1–25:26, April 2015.

[95] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. Anrl: Attributed network representation learning via deep neural networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial*

Intelligence, IJCAI-18, pages 3155–3161. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[96] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, Feb 2017.

[97] Qingyuan Zhao, Murat A. Erdogdu, Hera Y. He, Anand Rajaraman, and Jure Leskovec. SEISMIC: A self-exciting point process model for predicting tweet popularity. In *Proc. of KDD'15*. ACM, 2015.