UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Network Representation Learning with Attributes and Heterogeneity

Nasrullah Sheikh

Advisor

Prof. Alberto Montresor

Università degli Studi di Trento

March 2019

Abstract

Network Representation Learning (NRL) aims at learning a low-dimensional latent representation of nodes in a graph while preserving the graph information. The learned representation enables to easily and efficiently perform various machine learning tasks. Graphs are often associated with diverse and rich information such as attributes that play an important role in the formation of the network. Thus, it is imperative to exploit this information to complement the structure information and learn a better representation. This requires designing effective models which jointly leverage structure and attribute information. In case of a heterogeneous network, NRL methods should preserve the different relation types.

Towards this goal, this thesis proposes two models to learn a representation of attributed graphs and one model for learning representation in a heterogeneous network. In general, our approach is based on appropriately modeling the relation between graphs and attributes on one hand, between heterogeneous nodes on the other, executing a large collection of random walks over such graphs, and then applying off-the-shelf learning techniques to the data obtained from the walks. All our contributions are evaluated against a large number of state-of-the-art algorithms, on several well-known datasets, obtaining better results.

Keywords

[Graph Embedding, Attributed and Heterogeneous Graphs, Unsupervised Learning]

Dedicated to my parents.

Acknowledgment

Firstly, I would like to express my gratitude to my supervisor, Prof. Alberto Montresor for his advice and unflagging support over the course of the last three years. I am also grateful for his trust in me and for giving me the freedom to explore my research topic that helped me to accomplish this Ph.D. thesis.

Secondly, I would like to express my gratitude to my collaborators: Dr. Zekarias Kefato, Cristian Consonni, Dr. Amira Soliman, Dr. Leila Bhari, and Prof. Sarunas Girdzijauskas. I especially thank Zekarias Kefato for the extensive collaboration, ideas, and discussions. I thank Cristian Consonni, our Cricca research group member, for his help, insights, feedback and being there whenever needed. I would also like to thank my friends and colleagues at DISI, especially Dr. Kashif Ahmad, Dr. Maqsood Ahmad, Dr. Attaullah Buriro, Sudipan Saha, and Rajen Chatterjee for all scientific discussions and fun. Also, special thanks go to my friends in Kashmir: Dr. Firdous Ahmad, Zamir Ashraf, Attaullah, Dr. Mudasir, Umar, and Amjed. Further, I would like to thank ICT Secretariat and Ph.D. office, especially Andrea, Francesca and Roberta for their support in administrative matters.

I would also like to thank Prof. Sarunas for hosting me at the Royal Institute of Technology, Stockholm (KTH) for an internship. It was a valuable and rewarding learning experience, and a privilege to work with Prof. Sarunas, Leila, and Amira.

Next, I would like to take the opportunity to thank Dr. Dawood A. Khan, who has been my mentor since my bachelor's degree, for his invaluable support and discussions.

Finally, I thank and express my gratitude to my parents, sisters, and brother for their continuous support and encouragement throughout my years of study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graphs are ubiquitous: a large number of systems from diverse domains (social, biological, technological) can be represented as graphs. Examples include protein-protein networks, molecular structures, the World Wide Web, Online Social Network (OSN), power grids and communication networks. The entities present in the system are represented as nodes and the structural relationships between them are represented as edges. For example, in the case of an OSN, the entities are the users, and relationships such as friendship or follower-followee are the edges connecting them. Representing a system as a graph allows to exploit the expressive power of graphs and to obtain various insights about the system, such as pattern discovery.

Analyzing graphs through machine learning has a variety of application across different domains; for example, annotating proteins by role in protein-protein interaction networks [1, 2], classification of users and recommendation of new friends in online social networks [3–6], information diffusion in social networks [7, 8].

The performance of these applications largely depends on the features that are used to model the graph. A simple feature representation is an adjacency matrix that only captures the neighborhood information. Un-

fortunately, this representation does not encode the complex structural properties of the network such as higher-order proximities, which helps in the classification task. Moreover, the resulting representation is sparse and brings with it the curse of dimensionality; this deteriorates the performance of machine learning tasks. Furthermore, since graphs of interest tend to be really large, this representation makes machine learning tasks computationally expensive, hence not scalable to large graphs.

Earlier handcrafted features were used in machine learning tasks. These features include node degree, clustering coefficient, common neighbors, etc. The extraction of these features is time-consuming and expensive, in addition to their inflexibility to adjustment during learning.

To overcome these bottlenecks, Network Representation Learning (NRL) approaches have been proposed to encode and preserve the structural properties of a graph in a low-dimensional latent vector representation of nodes which is much smaller than the cardinality of the graph. NRL methods are built on a basic hypothesis: *"birds of same feather flock together"*, that is, similar vertices in the graph should be close to each other in the representation space. Following this, if two vertices are directly connected in the graph, the distance in the embedding space between their latent vector representation should be small. The learned representation makes it easy to perform various network analysis tasks, and machine learning tasks can also be directly applied by taking these learned embeddings as feature inputs as shown in Figure 1.1.

The success of NRL methods depends on their ability to exploit the different sources of information present in the graphs. The simplest and common source is the structural information, i.e the connectivity of nodes in the graph. The methods which use structural information focus on preserving the structural proximities of the graph. Graphs, however, are often associated with additional information such as attributes. The co-occurrence

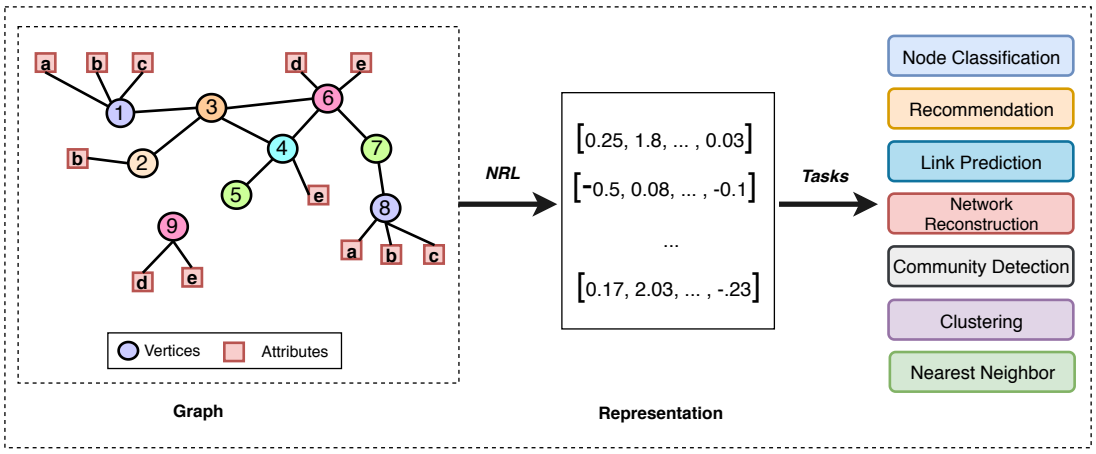

Figure 1.1: Overview of NRL Applications.

of attributes between nodes reinforces similarities between them, and thus alone structural proximities are not sufficient to capture the entire spectrum of similarities. For example, in the attributed graph shown in Figure 1.1, attributes are invaluable in cases where structural information is missing, or when structurally unrelated vertices have high attribute similarity. Vertex 9 is disconnected, but with the aid of attributes, it can have a representation similar to vertex 6. In the same way, vertices 1 and 8 are structurally far away from each other, but they have similar attributes. Therefore, the proximities in a graph can also be defined by their similarities in their attributes. By taking attribute proximities into consideration, representations of 1 and 8 will be close to each other. Moreover, the two sources of information can complement each other in learning, especially when our knowledge of one of them is only partial. Thus, it is of paramount importance to consider attribute similarities in order to learn precise embeddings. The challenges are compounded when such additional information is included, as the NRL methods have to encode and preserve both the structural and attribute proximities.

1.1 Machine Learning Tasks For Evaluation

We focused mainly on three machine learning tasks for the evaluation of our proposed approach. We briefly describe them in this section.

1.1.1 Node Classification

The *classification* task assigns a class label from a given set of classes to an unlabeled data based on a training dataset. When only two classes are possible, it is called *binary classification*. If there are more than two classes and each data point can have only one label, the task is called as *multi-class classification* whereas, when data instances can have more than one class label it is called as *multi-label classification*.

In this thesis, we leverage the structural and attribute information to learn a representation to effectively tackle node classification. We learn a representation and use it as feature vectors for the classifier. We use both multi-class and multi-label classification for node classification task depending on the dataset.

1.1.2 Link Prediction

The aim of link prediction is to predict the missing links or possible new connections based on the observed connections. For example, given an OSN, knowing the current connections (friendships), we want to predict the probability of an edge between two unconnected users; hence, we can recommend these two users to each other based on such probability. Other examples are academic networks such as co-author networks, in which it is interesting to know which two authors are highly likely to collaborate in the future. In the above examples, not only the connectivity, but also the attributes (e.g "likes" in an OSN and fields of interest in an academic network) may play a vital role in future connections.

In this thesis, to model predictions of future connections, we extract a residual graph by removing some edges and then learn a representation through our model. The removed edges serve as ground truth for evaluation, and the existence of an edge is computed from learned embeddings.

1.1.3 Nearest Neighbor Search

Nearest Neighbor (NN) is the proximity search algorithm which finds k-nearest data-points to a query data-point. In an attributed graph, the nearest nodes to a query node are those which are in close proximity, both in structure and attributes. For example, if two nodes, $B\{z_1, z_2, z_3\}, C\{z_4, z_5, z_6\}$ are connected directly to $A\{z_1, z_2, z_3\}$, it is obvious that node B is closer to A than node C, because node A has a higher attribute similarity to B rather than C.

In this work, we apply representation learning on an attributed graph in a way that unifies structural proximity and attribute proximity in a single d-dimensional vector space; thus, the similarity between any two given nodes can be computed by using a distance metric.

1.2 Thesis Overview and Contribution

The focus of this thesis is on network representation learning of attributed and heterogeneous graphs, building models that learn a better representation. We focus on three problem domains: attributed graph representation learning, heterogeneous graph representation learning, and cascade virality prediction.

For the sake of consistent reading of the thesis, Chapter 2 introduces some preliminaries and notations. Chapter 3 provides a detailed study of some models on which the proposed solutions are built. The state-of-the-art covering the major aspects of representation learning is discussed in

Chapter 4. The two models of Attributed Graph Embedding is discussed in Chapter 5 and 6. Chapter 7 presents a model for Heterogeneous Graph Embedding. A virality prediction model in OSN is presented in Chapter 8. Finally, we conclude this dissertation in Chapter 9. The contributions of this dissertation are the following:

1.2.1 Chapter 5: Joint Learning on Attributed Graphs

To learn a better representation, the two modalities (structure and attributes) need to be taken into account jointly to learn a representation, as they complement each other. We observe that a node has two contexts: a structural context and an attribute context. The structural context describes the nodes which are similar to a given node in terms of edge connections, whereas the attribute context of a node describes the nodes which are semantically similar to a given node. Therefore, a representation learning model needs to jointly optimize on these two distributions.

We propose GAT2VEC, an early fusion model that jointly learns a representation from structural contexts and attribute contexts. To obtain contexts from an attributed graph, we build two graphs: the structural one given by the nodes and their connections, and bipartite one, which connects vertices with the attributes. We employ random walks on both structural graph and bipartite graph to generate structural contexts and attribute contexts respectively. We then use the SKIPGRAM model to learn a representation from both contexts.

Through experimental evaluation, we showed that a joint learning model learns precise embeddings that preserve network proximities, through vertex classification, link prediction, and nearest neighbor results.

Publication Nasrullah Sheikh, Zekarias Kefato, and Alberto Montresor. GAT2VEC*: Representation learning for attributed graphs.* Computing 101(3):187-209. Springer, 2019.

1.2.2 Chapter 6: A Simple Approach to Learn Representation on Attributed Graphs

As the network structure is highly non-linear and sparse, shallow learning architectures learn poor representations. Furthermore, learning is even more challenging when attributed graphs are taken into considerations, as attributes themselves bring their own sparsity and non-linearity. This problem can be solved by designing deep neural networks which handle non-linearity and sparsity in both structure and attributes, but this adds more complexity. To model the proximities in an attributed graph, existing proposed methods use computationally expensive pre-processing–such as sampling–which limits their scalability.

We propose a simple enhanced autoencoder model (SAGE2VEC) that handles the non-linearity and sparsity of both the structure and attributes, without the need of computationally expensive preprocessing. We train the model on the network structure and optimize it on both the vertex neighborhood and the attributes. The experimental evaluation on vertex classification, link prediction, and network reconstruction tasks shows that our simple model is good at handling non-linearity and sparsity, while preserving the proximities.

Publication Nasrullah Sheikh, Zekarias T. Kefato, and Alberto Montresor. *A Simple Approach to Attributed Graph Embedding via Enhanced Autoencoder*. Submitted to Data Science and Advanced Analytics 2019.

1.2.3 Chapter 7: Heterogeneous Information Network Representation Learning

A heterogeneous network has multiple types of nodes and relationships, and each relationship has different semantics. The homogeneous network embedding methods cannot be applied to heterogeneous networks because

their sampling methods, such as random walks, are based on a homogeneous distribution of nodes and edges. Therefore, network representation learning approaches need to explicitly take care of different node types and relationships while sampling, such that the learned embeddings preserve the network properties.

In this thesis, we propose a relation-specific short random walk for generating sequences which represent the contextual relationships between nodes. We obtain a corpus of sequences by performing multiple random walks on each relation. The corpus of a sequence is analogous to sentences of a document, and we train a 1D-CNN for learning an embedding of nodes.

The preliminary results show that our proposed approach HetNet2Vec performs well and there is a lot of space for improvement.

Publication Nasrullah Sheikh, Zekarias T. Kefato, and Alberto Montresor. *Semi-supervised heterogeneous information network embedding for node classification using 1D-CNN*. In Proc. of the Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS 2018), pages 177–181. IEEE, October 2018.

1.2.4 Chapter 8: Predicting Virality of Cascades

This chapter slightly deviates from the main thesis topic as it deals with cascade virality prediction. The chapter is presented to show that the network agnostic approaches can be developed for drawing insights from a network system.

In Online Social Networks, it is common to see 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 Twitter account on February 6$^{\text{th}}$, 2018,

has been retweeted more than 75k times within the same day of posting[1]. Such diffusion events are called *viral cascades*. Predicting cascades virality is vital for different applications, for example to forecast trends and rumor break-outs [8]. However, it is challenging to effectively predict the virality of such kinds of events as early as possible, especially when little supporting information is available. Many research works have dedicated effort and attention to the prediction of content popularity with the focus of achieving good predictions in the shortest possible time, with the least information possible about the underlying network structure.

The diffusion of content on OSN happens through the underlying user connectivity graph, which plays an important role in determining the virality of content. Therefore, for the prediction of virality, it is necessary to take the properties of the underlying graph into account along with the cascade of information diffusion. The major challenge is to obtain the underlying social graph due to privacy concerns and the cost of mining such data. Thus, it is imperative to design virality prediction algorithms that are network oblivious but at the same time effective in prediction.

We propose CAS2VEC, a network-agnostic approach that uses explicit information available in cascades with the premise that the reaction time between two events is a sufficient indicator to predict the virality. The reaction time in cascades enables us to model it as a time series where each element is a discretized reaction time. Noticing that the distribution of reaction time is similar to the distribution of words in a language, we have applied CNN model from Natural Language Processing (NLP) to train our prediction model. The results from experimental evaluation testify our premise of modeling cascades as a time series and our model performs well in predicting the virality.

The organization of this chapter is as follows: Section 8.1 describes the

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

state-of-the-art, while Section 8.2 provides some definitions. The model for virality prediction is described in Section 8.3, followed by experimental setup and evaluation in Section 8.4.

Publication 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 International Conference on Social Networks Analysis, Management and Security (SNAMS 2018), pages 72–79. IEEE, October 2018 **Best Paper Award**.

1.3 Publication List

The publications which contributed to this thesis are the following:

1. Nasrullah Sheikh, Zekarias Kefato, and Alberto Montresor. *"gat2vec - Representation Learning for Attributed Graphs"*. In Journal of Computing. Springer, May 2018.

2. Nasrullah Sheikh, Zekarias T. Kefato, and Alberto Montresor. *A Simple Approach to Attributed Graph Embedding via Enhanced Autoencoder* Data Science and Advanced Analytics 2019 (under review)

3. Nasrullah Sheikh, Zekarias T. Kefato, and Alberto Montresor. *"Semi-Supervised Heterogeneous Information Network Embedding for Node Classification using 1D-CNN"*. In Proc. of the First International Workshop on Deep and Transfer Learning (collocated with SNAM). DTL '18. Oct. 2018.

4. Zekarias T. Kefato, Nasrullah Sheikh, et al. *"CAS2VEC: Network Agnostic Cascade Prediction in Online Social Networks"*. In Proc. of

the Fifth International Conference on Social Network Analysis, Management and Security. SNAM'18 Oct. 2018. **Best Paper Award**.

Other Contributions During the research period, I collaborated on other research works which resulted in the following publications:

1. Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. *"REFINE: Representation Learning from Diffusion Events"*. In Proc. of the Fourth International Conference on Machine Learning, Optimization and Data Science. LOD'18. Sept. 2018

2. Zekarias T. Kefato, Nasrullah Sheikh et al. *"CaTS: Network Agnostic Virality Prediction Model to Aid Rumour Detection"*. In Proc. of the Second International Workshop on Rumours and Deception in Social Media (Collocated with CIKM). RDSM'18. Oct. 2018.

3. Zekarias T. Kefato, Nasrullah Sheikh, and Alberto Montresor. *"MINERAL: Multi-modal Network Representation Learning"*. In Proc. of the Third International Conference on Machine Learning, Optimization and Big Data.MOD'17. ACM, Sept. 2017.

4. 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, Aug. 2017.

Chapter 2

Preliminaries

In this chapter, we introduce the various notations and definitions used in this thesis.

2.1 Notations

Scalars are represented with lower case letters, sets with uppercase letters. Matrices and vectors are denoted with uppercase and lowercase boldface letters, respectively. The main symbols used in this thesis are given in Table 2.1.

2.2 Definitions

Definition 1. *[**Attributed Graph**] Let $G = (V, E, \mathbf{M}, \mathbf{Z})$ be an attributed graph, where V is a set of n nodes, E a set of edges, $\mathbf{M}^{n \times n}$ is the adjacency matrix representation of the edges and $\mathbf{Z}^{n \times z}$ is the feature matrix representing z features associated with nodes.*

Let $\mathbf{M}_i$ be the i^{th} row vector and let $\mathbf{M}_{ij} \neq 0$ if there is an edge between the i^{th} and j^{th} vertex. Let $\mathbf{Z}_i$ be the feature vector of the i^{th} vertex, where $\mathbf{Z}_{iu} > 0$ if the i^{th} vertex has the u^{th} attribute associated with it.

Notation	Definition
n	number of nodes
z	number of features
d	embedding dimension
$\mathbf{M} \in \mathbb{R}^{n \times n}$	Adjacency matrix
$\mathbf{Z} \in \mathbb{R}^{n \times z}$	Feature matrix
$\mathbf{M}_i$ *or* $\mathbf{M}[i]$	Row vector of matrix $\mathbf{M}$
$\mathbf{M}_{ij}$ *or* $\mathbf{M}[i,j]$	Scalar value of $\mathbf{M}$
$\mathbf{m}_i$ *or* $\mathbf{m}[i]$	Scalar element of vector $\mathbf{m}$
$\Phi(.)$	function which return a $\mathbb{R}^d$-dimensional embedding of a vertex
$\|\|.\|\|_2$	l_2 norm of a vector
$\|\|.\|\|_F^2$	Forbenius norm of a matrix
$\odot$	Element wise multiplication

Table 2.1: Notations used in the thesis

For an *unweighted graph*, $\mathbf{M}_{ij} = 1$ in case there is an edge between vertex i and j. In case of a *weighted graph*, $\mathbf{M}_{ij} \neq 0$ is the weight of an edge between vertex i and j. If there is no edge, then $\mathbf{M}_{ij} = 0$ for both types of graphs.

Definition 2 (**First-Order Proximity[9]**). *The first-order proximity between two nodes i and j is determined by $|\mathbf{M}_{ij}| > 0$, that is, these two vertices are directly connected. It is also called as local pairwise proximity.*

Definition 3 (**Second-Order Proximity[9]**). *The second-order proximity between i^{th} and j^{th} node of graph G is determined by the similarity between their common neighbors i.e. similarity between $\mathbf{M}_i$ and $\mathbf{M}_j$.*

Definition 4 (**Higher-Order Proximity**). *Let $\hat{\mathbf{M}}$ be a one-step transition probability of $\mathbf{M}$ and $\mathbf{M}' = \hat{\mathbf{M}} + \hat{\mathbf{M}}^2 + \cdots + \hat{\mathbf{M}}^k$ is the k-step transition probability between each pair of vertex. The high-order proximity between i^{th} and j^{th} is determined by the similarity between $\mathbf{M}'_\mathbf{i}$ and $\mathbf{M}'_\mathbf{j}$.*

Definition 5 (**Context**). *Let $V = \{v_1, v_2, \cdots, v_n\}$ be the set of words (vocabulary) in a sentence (sequence) of size n; let k be the size of the*

context window. The context of a word v is c words before and after it, i.e $\{v_{i-k}, \cdots, v_{i-1}, v_{i+1}, \cdots v_{i_k}\}$.

Definition 6 (**Heterogeneous Graph**). *A* Heterogeneous Information Network *(HIN) is a graph $G_h = (V, E, f, g)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, $f : V \to T_V$ is a function mapping each node $v \in V$ to one node type in T_V, and $g : E \to T_E$ is a function mapping each edge $e \in E$ to one edge type in T_E.*

2.3 Evaluation Metrics

The embedding methods are evaluated using different machine learning tasks such as node classification and link prediction. In this section, we will give brief details about the metrics used to evaluate the tasks.

Definition 7. ***Precision*** *is the fraction of samples correctly classified as positive over all positively classified samples. Mathematically, it is defined below:*

$$pr = \frac{Tp}{Tp + Fp} \tag{2.1}$$

Definition 8. ***Recall*** *is the fraction of samples correctly classified as positive over all correct classifications as described below:*

$$rc = \frac{Tp}{Tp + Fn} \tag{2.2}$$

where in Equation 2.1 and 2.2, Tp, Fp, and Fn are *true-positives, false-positives, and false-negatives* respectively.

For an effective evaluation of the model, both precision and recall are necessary. Since they are antagonistic to each other, therefore, maximizing the combination of them gives an optimal solution, called F1-Score. It is computed as:

$$\text{F1-Score} = 2 \cdot \frac{pr \times rc}{pr + rc} \tag{2.3}$$

The general case of F-measure is given as:

$$\text{F}_\beta\text{-SCORE} = (1 + \beta^2) \cdot \frac{pr \times rc}{\beta^2 \times (pr + rc)} \tag{2.4}$$

In the case of multiclass and multilabel classification F1-SCORE is averaged due to the presence of independent classes. MICRO-F1 and MACRO-F1 are two statistics calculated in this case. MACRO-F1 gives the equal weight to each class and the metrics are calculated for each class. Let $p_r(.)$ and $r_c(.)$ be the functions that return the *precision* and *recall* for each class respectively. The average *precision* (pr_{avg}) and *recall* (rc_{avg}) are calculated as:

$$pr_{avg} = \frac{\sum_{c \in C} p_r(c)}{|C|}$$

$$rc_{avg} = \frac{\sum_{c \in C} r_c(c)}{|C|}$$

where C is the set of classes.

MACRO-F1 is given is harmonic mean of average *precision* and *recall* calculated as:

$$\text{MACRO-F1} = 2 \times \frac{pr_{avg} \times rc_{avg}}{pr_{avg} + rc_{avg}} \tag{2.5}$$

In MICRO-F1 average, we sum the scores of different class and calculate the overall precision and recall. Let $tp_s(.)$, $fp_s(.)$, and $fn_s(.)$ be the functions that return the true positives, false positives and false negatives for a given class, respectively. MICRO-F1 is calculated as:

$$pr_a = \sum_{c \in C} \frac{tp_s(c)}{tp_s(c) + fp_s(c)} \tag{2.6}$$

$$rc_a = \sum_{c \in C} \frac{tp_s(c)}{tp_s(c) + fn_s(c)} \tag{2.7}$$

MICRO-F1 is the harmonic mean of pr_a and rc_a as given below:

$$\text{MICRO-F1} = 2 \times \frac{pr_a \times rc_a}{pr_a + rc_a} \tag{2.8}$$

Chapter 3

Background

This chapter provides the necessary and sufficient details of the methods on which the work in the thesis is built upon. The sections 3.1, 3.2, and 3.3 are the foundations for the solutions proposed for Chapters 5, 6, 7 and 8.

3.1 SkipGram

Representing the words of text in a continuous vector representation is a very crucial problem in Natural Language Processing (NLP), as they significantly simplify and improve NLP applications. The continuous vector representation preserves the semantic meaning of words and overcomes sparsity. Various models that have been proposed earlier were limited by their lack of scalability i.e., they were significantly computationally expensive in training [10–12]. These model were trained on small corpora, thus the learned representations were of low quality. Mikolov et al. proposed a distributed word representation learning model called WORD2VEC that is scalable and preserves the semantic meanings of the words [13]. The preservation of the semantic properties is demonstrated through various examples, such as the vectors of synonyms of a word tend to be close in representation space. Another fascinating example is from the analogy, *"Woman is to queen as man is to king"*; it turns out that this is true via

algebraic operation as well: $\Phi(queen) - \Phi(woman) + \Phi(man) \approx \Phi(king)$, where $\Phi(.)$ returns the embedding of the given word.

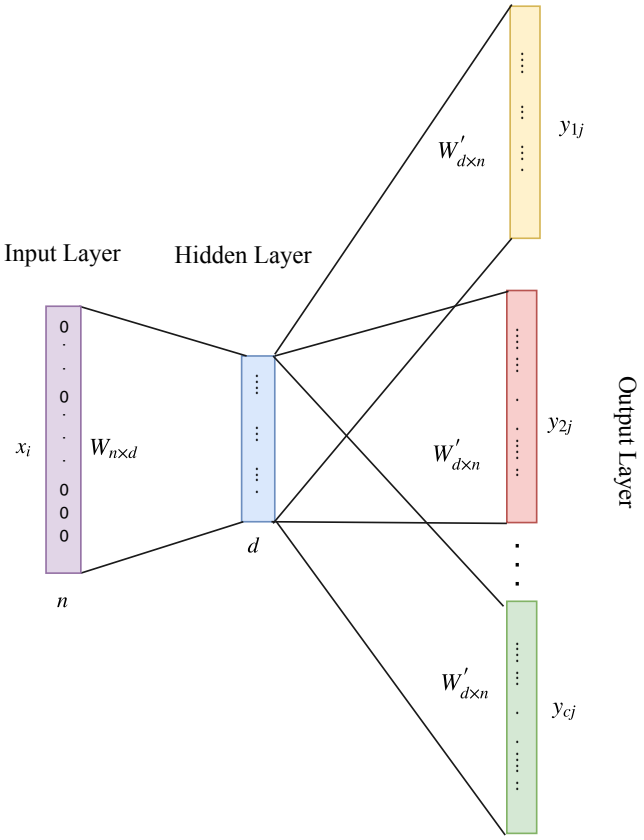

Figure 3.1: Architecture of SkipGram model

Word2Vec proposed two architectures, Continuous Bag-of-words (CBOW) and SkipGram for learning representation of words. In the CBOW model, the occurrence of a word is predicted from a given set of context words. The model is trained through a log-linear classifier which tries to correctly predict a middle word from a set of context words. The experiments show that this model does not perform better than SkipGram. Therefore, we skip the details of this model and we focus on SkipGram. The architecture of SkipGram is shown in Figure 3.1. Let $V = \{v_1, v_2, \cdots, v_n\}$ be the set of words (vocabulary) of size n, C be the set of all contexts, and T be the set all word and context pairs $(v, c) \in T$; i.e for each word v, c

are all contexts associated with it. For a given word v and its contexts c, the SkipGram model considers the conditional probabilities $p(c|v)$. The probability is calculated using soft-max as given below:

$$p(c|v) = \frac{e^{\Phi(c).\Phi(v)}}{\sum_{c' \in C} e^{\Phi(c').\Phi(v)}} \tag{3.1}$$

The goal is find a parameter θ such that probability $p(c|v;\theta)$ is maximized as:

$$\arg\max_{\theta} \prod_{(c,v)\in T} p(c|v;\theta) \tag{3.2}$$

Applying the logarithm we get:

$$\arg\max_{\theta} \prod_{(c,v)\in T} p(c|v;\theta) = \sum_{(c,v)\in T} \left(log\ e^{\Phi(c).\Phi(v)} - log \sum_{c' \in C} e^{\Phi(c').\Phi(v)} \right) \tag{3.3}$$

Since summation over all other contexts $c^{'}$ in the above equation makes it computationally expensive as there can be thousand of other contexts, the soft-max is replaced with the *hierarchical softmax*.

Hierarchical softmax is an approximation to softmax proposed by Morin and Bengio [14]. It is a complete binary tree where the words are the leaf nodes and the root is the given word. Thus, the prediction task is to find a path that maximizes the probability for the given word.

3.2 Autoencoder

An autoencoder is an unsupervised neural network model that learns a representation of the input data. Traditionally, autoencoders have been used for dimensionality reduction and feature learning [15–17]; due to recent advancements in neural networks, they are also used in generative models. An autoencoder consists of two parts: an *encoder function* which

compresses the input data $\mathbf{y} = f(\mathbf{m})$ and a *decoder function*, $\tilde{\mathbf{m}} = g(\mathbf{y})$ tries to reconstruct the input as close as possible. $\mathbf{m}$ is an input vector, $\mathbf{y}$ is the latent representation, and $\tilde{\mathbf{m}}$ is the reconstruction. In other words, the autoencoder tries to learn an identity function such that $\mathbf{m} \approx \tilde{\mathbf{m}}$. The autoencoder functions are constrained so that they are not able to copy the input exactly. This is achieved by forcing the autoencoder to prioritize to learn only useful properties of the input which is achieved by constraining $\mathbf{y}$ to a smaller dimension than $\mathbf{m}$. A possible architecture of an autoencoder with one hidden layer is shown in Figure 3.2. The two red circles in the decoder signify the error in reconstruction. Mathematically, encoders and decoders are described by Equation 3.4 and 3.5, respectively.

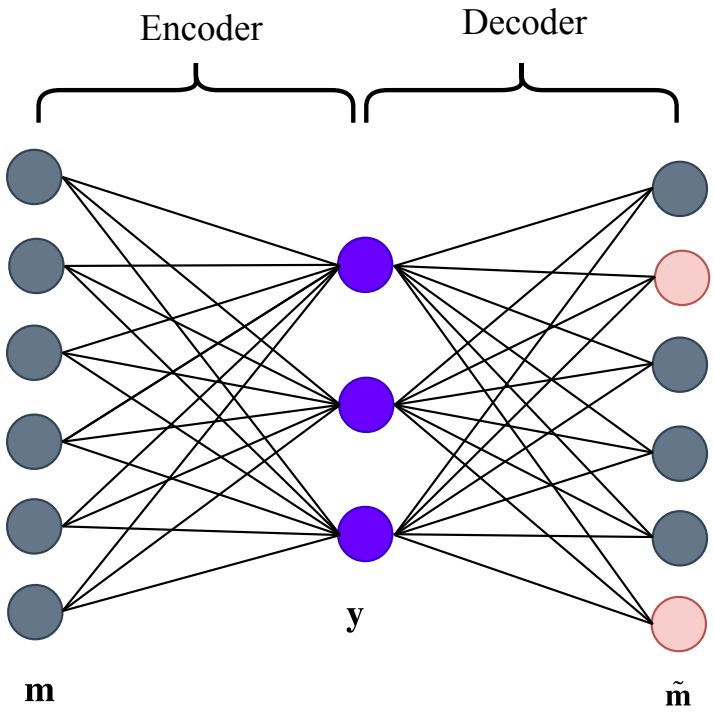

Figure 3.2: Architecture of Autoencoder model.

$$\mathbf{y} = f(\mathbf{W}.\mathbf{m} + b) \tag{3.4}$$

$$\tilde{\mathbf{m}} = g(\tilde{\mathbf{W}}.\mathbf{y} + b) \tag{3.5}$$

Here $\mathbf{W}, \tilde{\mathbf{W}}$ are weight matrices for encoder and decoder respectively and b is the bias term.

The learning objective of an one-layer autoencoder is described by the minimizing loss function as given under:

$$\mathcal{L} = \mathcal{E}(\mathbf{m}, \tilde{\mathbf{m}}), \tag{3.6}$$

where $\mathcal{E}$ is the loss function that penalizes $\tilde{\mathbf{m}}$ for being dissimilar to $\mathbf{m}$. Mean Square Error (MSE) and binary cross entropy are some examples of loss functions used. The MSE loss is given as below:

$$\mathcal{L} = \|\mathbf{m} - \tilde{\mathbf{m}}\|_2^2 \tag{3.7}$$

In general, for a deep autoencoder with L layers and $\mathbf{m}$ as input, the encoder function can be described as:

$$\begin{aligned} \mathbf{y}^{(1)} &= f(\mathbf{W}^{(1)}\mathbf{m} + b^{(1)}) \\ \mathbf{y}^{(l)} &= f(\mathbf{W}^{(l)}\mathbf{y}^{(l-1)} + b^{(l)}), \quad l = 2, \cdots, L \end{aligned} \tag{3.8}$$

$\mathbf{y}^l$ is the latent representation of $\mathbf{m}$ at the l^{th} layer; L is the number of layers; $\mathbf{W}^{(l)}$ is the weight matrix and $b^{(l)}$ is the bias at the l^{th} layer. The decoder is the reverse of the encoder; thus, it takes $\mathbf{y}^{(l)}$ as input and produces the reconstruction $\tilde{\mathbf{m}}$.

An autoencoder learns an embedding in the same subspace as PCA when the decoder function g is linear and the error function is MSE. In case, f and g are both non-linear functions; thus, learned embeddings are nonlinear generalizations of PCA.

3.3 Convolutional Neural Network

Convolutional Neural Network is a deep neural network which operate on grid-like data such as images and sequences [18]. CNN extracts local features by exploiting shift-invariance, local connectivity, and compositionality of the data. A CNN consists of three components: the convolutional layer, the pooling layer, and the fully-connected layer. An example architecture[1] of CNN is given in Figure 3.3. The architecture takes an image with three channels (RGB) as input and applies two convolutions and max-pooling operations.

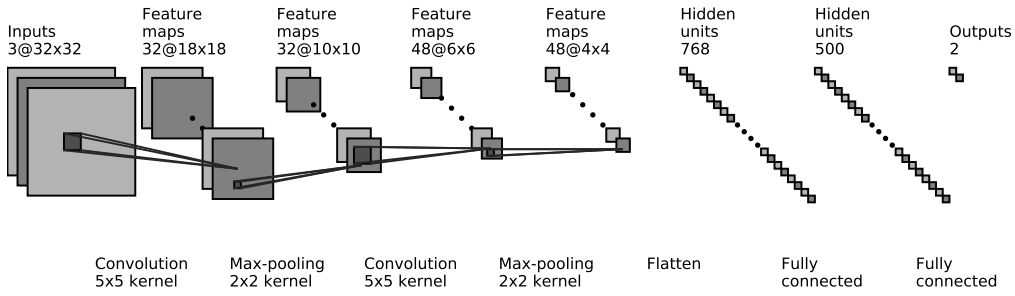

Figure 3.3: Architecture of Convolutional Neural Network

Convolutional Layer This layer applies the convolution operation to the input; the output from the convolution is called *feature map*. The convolution operation is performed through filters, also referred as *kernels*; the areas on which convolution is applied is called *receptive field*. Kernels enable the CNN to extract high level features which are shift and positional invariant. Let $\mathbf{I}$ be an input 3-channel(RBG) image of size $n \times n$; kernel $\mathbf{K}$ has a size $h \times w$ and the depth of the filter is equal to number of channels.

[1]Generated using `https://github.com/gwding/draw_convnet`

The i, j-th feature from a receptive field is computed as as:

$$\mathbf{C}[i,j] = \sum_{p}\sum_{q} \mathbf{I}[i-p, j-q] \cdot \mathbf{K}[p,q] \tag{3.9}$$

The convolution performs an element-wise multiplication with the input. The filter is slid horizontally and vertically along the image, and the slide is controlled by parameter s. In Figure 3.3, the first convolution layer has a filter of $5 \times 5 \times 3$, and $s = 1$, thereby producing a feature map of 28×28. To preserve the spatial dimensions, a number of filters are applied. In the example, we have 32 filters producing 32 feature maps; on each of them, an activation function is applied. The weights of filters are trained through the backpropagation algorithm.

Pooling is a downsampling process to further reduce the size of the input by computing a summary statistic of nearby points. Pooling can be done through various methods such as *max-pooling* and *average-pooling*. In *max-pooling*, a maximum value is returned from a patch of the image and the rest are discarded. In *average-pooling*, an average value is returned from a patch.

Fully Connected Layer The output from the pooling is flattened and fetched to a fully connected layer component. The number of FC-layers vary depending on the complexity of the data and the model. The result from the FC-layer can be used for various machine learning tasks such as classification.

Chapter 4

State-of-the-art

The objective of this chapter is to provide an overall overview of the methods designed to learn a representation of nodes in a graph. This chapter will outline an evolution of techniques–from simple to complex–exploiting different modalities. Broadly, we classify the NRL methods in two categories: *homogeneous* and *heterogeneous* network embedding, depending on the type of nodes and relationships present in a graph. In homogeneous NRL methods, the nodes and the edges are of a single type, whereas in the case of heterogeneous NRL methods nodes and edges are of different types.

4.1 Homogeneous Network Embedding

In this section, we describe the various methods proposed to learn a representation. These methods use various different sources of information for learning. Broadly speaking, homogeneous network embedding methods are divided into two groups: *plain network embedding* and *attributed network embedding*. To learn a representation, the former methods use only the structure of the network, whereas the latter methods use both the network structure and the attribute information associated with the nodes.

4.1.1 Plain Network Embedding

Matrix Factorization Methods

Earlier methods of NRL focused on matrix-factorization techniques to embed a high dimensional network representation, often an adjacency matrix into a low dimensional space. Different algorithms apply different factorization strategies, such as Laplacian Eigenmaps (LE)[19], Modularized-Non-Negative Matrix Factorization (M-NMF)[20] and others [21–25]. The approach proposed by Belkin and Niyogi uses an adjacency matrix to construct a weighted input matrix such that two nodes are connected if they are close to each other [19]. The "closeness" can be quantified by various methods; for example if they are among k-nearest neighbors. Then they compute the eigenvalues and eigenvectors on the weighted input matrix. The top-d eigenvectors correspond to the embedding of the input. Other approaches also use eigenvectors such as [21, 22]. Tang et al. use modularity matrix of the graph and top-d eigenvectors represent the learned embedding [21]; whereas in SocioDim they use normalized Laplacian matrix to generate an embedding from d-smallest eigenvectors [22].

The above approaches work on a single input matrix. Cao et al. argue that preservation of k-step relational information leads to a better representation learning [23]. Hence, they propose GraRep which captures k-step different relational information. GraRep uses k-hop normalized adjacency matrix, and for each k learns an embedding using matrix factorization using SVD which captures local information. The final embedding is obtained by concatenating these k different embeddings.

Neural Network Methods

Due to recent advances in neural networks, NRL has drawn a lot of attention from researchers, proposing approaches from shallow [26] to deep

models [27]. Inspired by NLP, Perozzi et al. observed that the distribution of words in documents is similar to the distribution of vertex sequences obtained through random walks on a graph [26]. Each vertex is analogous to a word in a document. They proposed DeepWalk, which adopts the SkipGram [13] model to learn a representation of vertices in the graph. DeepWalk exploits truncated short random walks to generate a vertex sequence which preserves the neighborhood structure of nodes. A number of random walks are performed on each vertex to generate a corpus of sequences. For each walk sequence, DeepWalk aims to maximize the probability of occurrence of a node in a given context of size w.

Various other random-walk based approaches have been proposed [28–31]. These approaches apply different random-walk sampling methods or optimization techniques to learn an embedding. Node2Vec proposed by Grover et al. employs biased random walks to explore the diverse neighborhood of network [28]. It uses two random walk parameters p and q; parameter p controls the probability of revisiting a vertex whereas q controls the probability of visiting a node's one-hop neighborhood. Hence, these hyperparameters help in smoothly interpolating between depth-first or breadth-first sampling. Furthermore, Node2Vec employs negative sampling in contrast to hierarchical softmax in DeepWalk. These approaches preserve the higher order proximities. A node classification specific method, Discriminative Deep Random Walk (DDRW) proposed by Juzheng et al. captures the global network structure and are discriminative to network classification task [30].

Structural and Neighborhood Similarity (SNS) uses two aspects in a graph, neighbor information, and local subgraphs similarity to learn an embedding [29]. In addition to employing random walk sampling to capture the global information, the method uses graph mining techniques to capture the structural equivalence of nodes in subgraphs. Another aspect

of network embedding is to capture and preserve the structural identity of nodes in the graphs. In this line, STRUCT2VEC is proposed which employs a biased random walk to generate the context of nodes in a multilayer graph to preserve structural identities of nodes [31].

An edge sampling based method called LINE captures first-order and second-order proximities to learn an embedding [9]. The edges are sampled with the probabilities proportional to their weights. LINE models first-order proximity by joint probability distribution between vertices, and the empirical proximity distribution which is calculated by normalizing the weight of an edge with a sum of all edge weights. The first-order proximity is preserved by minimizing these two joint probability distributions. In case of second-order proximity, two vertices which have similar contexts are considered close. For each edge, LINE calculates a conditional distribution of a vertex in a given context and also an empirical distribution. The second-order proximity is also preserved by minimizing these two joint probability distributions. The overall objective of LINE is to jointly minimize these two objectives.

The above-mentioned models are shallow, thus they do not learn good quality representations. As pointed out in various works, the underlying networks are complex structures [32], highly non-linear [33], and sparse [26]. Therefore, it requires deep models to overcome these challenges to learn a good quality representation. Wang et al. proposed Structural Deep Network Embedding (SDNE) to preserve the structure, handle sparsity, and non-linearity of the network [27]. SDNE takes adjacency matrix as input and employs a semi-supervised deep autoencoder model which exploits first-order and second-order proximity to preserve structural properties of a graph. It adopts the idea of Laplacian Eigenmaps (LE) for preserving first-order proximity [19]. To address the sparsity issues, reconstruction errors of non-zero elements are penalized more than zero elements. In another

method, DNGR, a stacked deep denoising autoencoder is used to learn an embedding [34]. The proposed approach captures structural information through random surfer model and generates probabilistic co-occurrence matrix. From it, *positive point-wise mutual information* (PPMI) matrix is generated which is fetched into denoising autoencoder to learn an embedding.

Adapting Deep Neural Network models to work on graphs has led to a new domain called Graph Neural Network (GNN). Gori et al. were the first to apply neural networks on graphs [35] followed by an extension by Scarselli et al. [36]. These approaches learn an embedding of nodes by propagating neighbor information through an iterative architecture which is computationally expensive. The tremendous success of Convolutional Neural Network (CNN) in computer vision and NLP has motivated researchers to apply these methods on graph data which led to Graph Convolutional Networks (GCN). Various GCNs based on spectral graph theory have been proposed [37–41]. Bruna et al. were the first to formulate CNN on graphs [37]. The authors proposed two constructions to use CNN in graph data: spatial domain-based on multi-scale clustering and spectral domain based on graph Laplacian. In spatial convolution, the connectivity matrix is a k-hop adjacency, the features are summed up from the neighbors and the convolution layer is a fully connected layer working on sparse connectivity matrix of k-hops.

4.1.2 Attributed Network Embedding

Matrix Factorization Methods

In this class, the NRL methods apply matrix factorization techniques on the adjacency matrix of the graph and the context matrix to learn a representation [42–45]. Text Associated DeepWalk (TADW) incorporates text

features associated with vertices through matrix factorization [44]. The authors show that the DeepWalk is equivalent to matrix factorization and the random walk of length t is equivalent to multiplying the adjacency matrix t times and then factorizing it. The factorization takes the text associated matrix into account. Given that some attributed graphs can be partially labeled as groups or community categories, then these labels can be helpful in learning a better representation. Huang et al. proposed a Label Informed Attributed Network Embedding (LANE), a semi-supervised approach which incorporates labels in addition to attributes in learning a representation [42]. LANE separately applies spectral methods on network structure, attribute matrix and labels matrix to learn a latent representation, and then these representations are projected in a unified space to obtain a final representation. Due to the large size of graphs and the associated attributes, the MF methods are limited due to scalability. To address this, the authors of LANE proposed Accelerated Attributed Network Embedding (AANE), which divides modeling and optimization into sub-problems which are solved in a distributed environment [43].

Neural Network Methods

Various approaches have been proposed that use deep neural networks to learn an embedding jointly from network structure and the attributes [46–53]. TriDNR is a semi-supervised approach which uses network structure, network attributes and partial labels to learn an embedding [48]. It employs a coupled model - one model exploits the inter-node relationships by using SkipGram on network structure, other one exploits vertex-node and label-word relationships by using Doc2Vec on node attributes and labels. The model uses a late fusion approach, i.e. the learning is performed independent of each other and the final embedding is obtained by a linear combination on two embeddings. The drawback with this approach

is that two different sources of information (structure and attributes) do not complement each other in learning. Another semi-supervised approach called Predictive Text Analytics (PTE) originally designed for text networks, can also be used for graphs with attributes [46]. The *word-word*, *word-document*, and *word-label* networks described in PTE can be translated to *node-node*, *node-attributes*, and *node-label* networks for attributed graphs.

Graph Convolutional Network (GCN) based approaches have also been proposed for attributed graphs. Kipf and Welling proposed a fast and scalable semi-supervised approach that uses first-order approximations of spectral graph convolutions [53]. This approach is transductive, i.e. trains on network structure, thus the model cannot generalize to unseen nodes. An inductive approach called GraphSAGE uses node features for training and generated embedding for the vertices [2]. The model trains a set *aggregator functions* that aggregate information from nodes at a given depth from the current node. Thus, each node is represented by the aggregation of its neighbors. The paper proposed three aggregation functions: *mean aggregator*, *LSTM aggregator* and *pooling aggregator*. This process is done iteratively for k steps and at each step, the latent representation is updated as per aggregator functions.

Graph autoencoder has also shown promising results in learning an embedding using GCN [52, 54]. Variational Graph Autoencoder (VGAE)[52] is an unsupervised approach which uses GCN encoder and a simple inner product decoder.

Gao et al. pointed out that not only network structures are highly non-linear, but also the attributes of the network are non-linear as well [49]. Therefore, they proposed Deep Attributed Network Embedding (DANE), which employs a deep autoencoder on both the structure and the attributes simultaneously. The proposed approach preserves the first-order proximity

in the network structure and the attributes, and the high-order proximity in the network structure only. Furthermore, the model maintains the consistency between two modalities, and a complementary representation is obtained by jointly optimizing on the joint distribution of two modalities. This requires sampling the vertices which are similar in attributes. The sampling strategy is quadratic which limits the scalability of DANE. Another approach, ANRL uses an enhanced deep autoencoder which preserves structural proximities and attributes affinities [50]. The model takes only attributes as input and reconstructs the attributes and neighborhood of the given node. The neighborhood reconstruction is optimized against ground truth which is prior generated through random walks. Most of the approaches only learn the representation of the nodes and not the attributes. A recent model called Co-embedding Attributed Networks (CAN) learns an embedding of both nodes and the attributes and use them in applications such as user profiling [51]. CAN employs a variational autoencoder and captures the affinities between nodes and attributes by projecting them in the same semantic space [55].

4.2 Heterogeneous Network Embedding

Some networks are by and large inherently heterogeneous. The NRL methods designed for a homogeneous network cannot be directly used for these networks as these methods discard the node and edge types and the learned embedding cannot preserve the properties of such a network. On this account, various approaches have been proposed [56, 56, 57, 57–60]. Yuxio et al. proposed Metapath2Vec, which uses meta-path based random walks to generate the heterogeneous neighborhood contexts [56] and introduced heterogeneous SkipGram model to learn an embedding. The meta-path based random walk uses a meta-path scheme which guides a random walker

to choose the next node in the walk, such that the semantics and structural correlations between different types of nodes are captured. The heterogeneous SkipGram model maximizes the probability of a node in heterogeneous contexts and for efficient optimization, it uses negative sampling approach [61]. In an extension approach called Metapath2Vec++, they use heterogeneous negative sampling which normalizes the softmax function with respect to the node type of the context. Hin2Vec is another meta-path based approach which learns the node representations as well as representations of the targeted relationships that are used to predict the target relationships between two nodes [58]. Hin2Vec is a single layer, feed-forward neural network that takes 3 inputs- two nodes and the relationship between them - and trains a binary classifier which predicts whether the two input nodes have a specific relationship or not. The cumbersome task is to sample the training data such that it covers as many node pairs and their relationships as possible. Moreover, the binary classifier also requires negative samples which are populated by replacing one of the nodes in the input data tuple. The problem with the meta-path based approach is that it requires a user to define the schema to guide random walks, and second, these approaches require a very large number of walks for adequate sampling. To overcome these drawbacks, Rana et al. proposed a Jump and Stay strategy to perform random walks which not only performs better with meta-path based learning methods but also speeds up the learning [62].

A deep network approaches Heterogeneous Network Embedding (HNE) considers a heterogeneous network having different types of nodes and contents such as images, text [57]. Thus leading to a network in which we can have *text-text* links, *image-image* links, and *image-text* links. Thus, HNE strives to learn a unified embedding which preserves the content and relational information. The neural network of HNE consists of three modules:

image-image, *image-text*, and *text-text* which are fed with pairwise data from the HIN. The *image-image* module employs a Convolutional Neural Network (CNN) for each image, *image-text* employs a CNN for image and Fully Connected (FC) layers for text, and FC layers for each text data input in *text-text* module. The weights are shared within and between the modules, and components are optimized through Stochastic Gradient Descent (SGC) for learning a unified embedding.

Chapter 5

Joint Learning on Attributed Graphs

The objective of network representation learning is to embed vertices in a low-dimensional space where the graph properties such as pairwise relationships between vertices and the structure of vertex local neighborhood are preserved. The properties between vertices can be captured through local information (e.g. followers, citations, friendship relations) or global information (e.g. h-hop neighborhood, community affiliation). The similarity of a vertex with respect to other vertices represents its contextual information. The contexts obtained using structural information of the network are called *structural contexts*.

Similarly to structural contexts, we can define the context of vertices in terms of attributes. The contextual information based on attributes defines the semantic relationship between vertices. For example, in a citation network, two papers having similar keywords share contextual information irrespective of their distance in structure. It is a challenging task, however, to generate attribute contexts from attributed graphs, in particular when the coverage of attributes is only partial, as in Figure 5.1. The labels of vertices are used in the classification task.

In this chapter, we introduce GAT2VEC, a framework that jointly learns from both the structure and the attributes of the network using a single

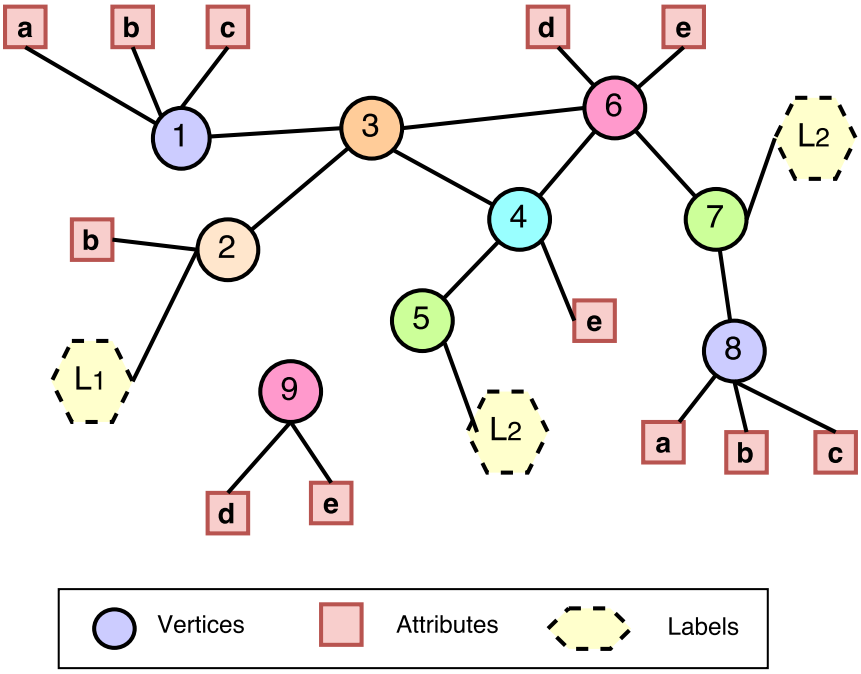

Figure 5.1: An example of a partially attributed graph.

neural layer. Structural contexts are obtained from the graph, preserving structural proximities; attribute contexts are obtained from a bipartite graph linking vertices and their attributes, preserving content proximities.

The proposed framework learns a representation in an unsupervised manner, scaling to large graphs, for both directed and undirected homogeneous graphs. Our approach is novel as it leverages multiple sources of information through early fusion and needs to optimize a single objective function. Furthermore, we will present a semi-supervised variant of GAT2VEC called GAT2VEC-WL, where labels are incorporated as attributes to enhance the learning of embeddings.

We empirically evaluated and validated our approach on vertex classification (multi-class & multi-label) and link prediction, on real-world attributed networks. The qualitative analysis from visualization and query task also validates our approach.

The rest of the chapter is organized as follows. In Section 5.1, we provide

the motivation and formally describe the problem. In Section 5.2, we present our proposed framework GAT2VEC. We describe the experimental setup and experimental results in Section 5.3.

5.1 Problem

We investigate the problem of integrating structural and attribute contextual information obtained from a partially attributed graph and employ a neural network model to jointly learn a representation of the vertices in a low-dimensional space. Informally, the problem can be described as follows:

Problem Given an attributed graph G as shown in Figure 5.1, we aim at learning a low-dimensional network representation $\Phi : V \rightarrow \mathbb{R}^d$, where $d \ll |V|$ is the dimension of the learned representation, such that the structural and attribute contextual informations are preserved.

The learned representation Φ is generic and can provide feature inputs for various machine learning tasks, such as classification and link prediction. Its validity is evaluated through such machine learning tasks; the precise figures of merit to be used are described in the respective Sections 5.3.4 and 5.3.5. For example, if the learned representation is able to classify vertices with high precision and recall (combined as F1 score), that means that the contextual information of vertices is well-preserved.

In this work, we consider G as a homogeneous, un-weighted and partially attributed graph. We evaluate our proposed approach on classification of vertices (multi-class and multi-label) and link prediction. We also perform a qualitative analysis (nearest-neighbor search and visualization).

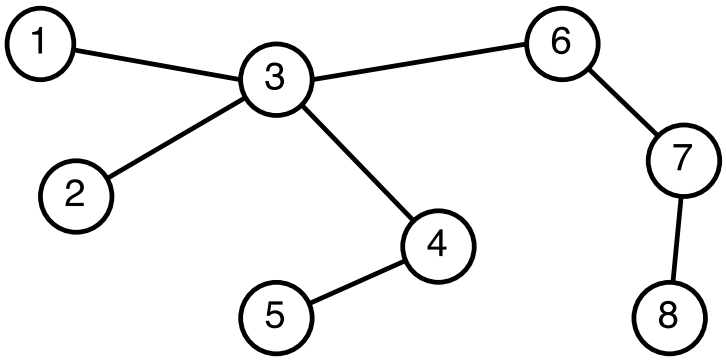

Figure 5.2: A graph depicting the structural relationships between vertices.

5.2 GAT2VEC Framework

In this section, we present a detailed description of our proposed framework GAT2VEC[1] which learns a network embedding from structural and attribute information. For each vertex, we obtain its structural and attribute contexts with respect to other vertices through random walks. Then, we integrate these two contexts to learn an embedding which preserves both structural and attribute proximities. The GAT2VEC method is outlined in Algorithm 1. Specifically, our framework consists of three stages:

- Network generation
- Random walks
- Representation learning

5.2.1 Network Generation

From the attributed graph, G, we obtain two graphs:

(1) A connected structural graph $G_s = (V_s, E)$, consisting of a subset $V_s \subseteq V$ of vertices that have connections included in the set E of edges. We refer to vertices in V_s as *structural vertices*. An edge

[1] `https://github.com/snash4/GAT2VEC`

$(p_s, q_s) \in E$ encodes a structural relationship between nodes as shown in Figure 5.2.

(2) A bipartite graph $G_a = (V_a, \mathbb{A}, E_a)$, consisting of (i) the subset of *content vertices* $V_a \subseteq V$ that are associated with attributes, (ii) the set of possible *attribute vertices* $\mathbb{A}$ derived from $\mathbf{Z}$ as given in the Definition 1 in Section 2.2 of Chapter 2, and (iii) the set of edges E_a connecting content vertices to the attribute vertices that are associated by function A:

$$V_a = \{v : A(v) \neq \emptyset\}$$
$$E_a = \{(v, a) : a \in A(v)\}$$

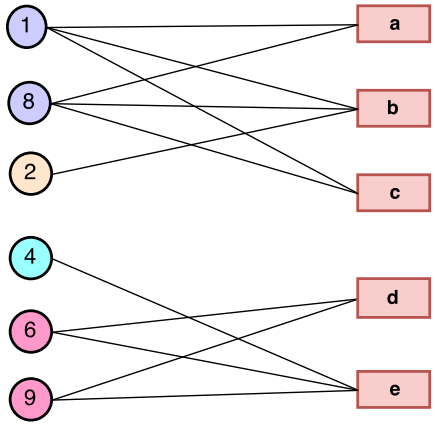

Figure 5.3: A bipartite graph between content nodes and attributes.

Proposition 1. *Two content vertices $u, v \in G_a$ are reachable only via attribute vertices.*

Following Proposition 1, the path between content vertices contains both content and attribute vertices. Therefore, the content vertices contained in the path have a contextual relationship because they are reachable via some attribute vertices. These content vertices form the attribute

contexts. The intuition behind our approach is "two entities are similar, if they are connected with similar objects". This phenomenon can be observed in many applications, e.g. in the bipartite "user-product" graphs, where "users" buy "products" and two users are similar if they buy similar products.

Such bipartite network structure has also been used in [46] which models a network between documents and the words included in them.

5.2.2 Random Walks

Informally speaking, similarity between entities could be measured in several ways. For example, it could be measured based on the distance of two nodes in the graphs. In the attributed graph of Figure 5.3, e.g., nodes 2 and 8 are both one attribute away from node 1, and thus we could say that they have equal similarity to 1; but there three paths connecting 1 and 8, while there are only two paths connecting 1 and 2; thus, 1 and 8 are more similar than 1 and 2.

While several approaches have been used [9, 26–28, 48], we adopt short random walks to obtain both the structural and the attribute contexts of vertices at the same time. The short random walks enable to effectively capture the contexts in which nodes have high similarity [26].

Random walks are performed on both G_s and G_a. The random walks over G_s capture the structural context. For each vertex, γ_s random walks of length λ_s are conducted to build a corpus R. This contextual information is used in the embedding, with the aim of maintaining the local and global structure information. We denote r_i as the i-th vertex in the random walk sequence $r \in R$.

For example, a random walk in the graph of Figure 5.2 could be the following: $r = (2, 3, 4, 3, 1)$, with length length 5, starting at vertex 2 and ending at vertex 1.

In the bipartite graph G_a, a random walk starts with a content vertex and jumps to other content vertices via attribute nodes. The attribute vertices act as bridges between content vertices and help in determining the contextual relationships among them, i.e. which content vertices are closely related. As we are interested in how often such vertices co-occur in random walks and not in which attributes have been traversed to connect them, we have omitted the attributes in our random walks. Thus, the walks contain only vertices from V_a.

Group of vertices that have high similarity in attributes are likely to appear frequently together in the random walks. Similar to G_s, we perform γ_a random walks of length λ_a and build a corpus W; we denote with w_j the j-th vertex of the random walk sequence $w \in W$.

For example, a random walk with attributes in the graph of Figure 5.3 could be the following: $[2, b, 1, c, 8, b, 2, b, 8]$. Since we are skipping attribute nodes in walks, therefore the corresponding walk is $w = [2, 1, 8, 2, 8]$, with length 5, starting from vertex 2 and terminating in vertex 8.

5.2.3 Representation Learning

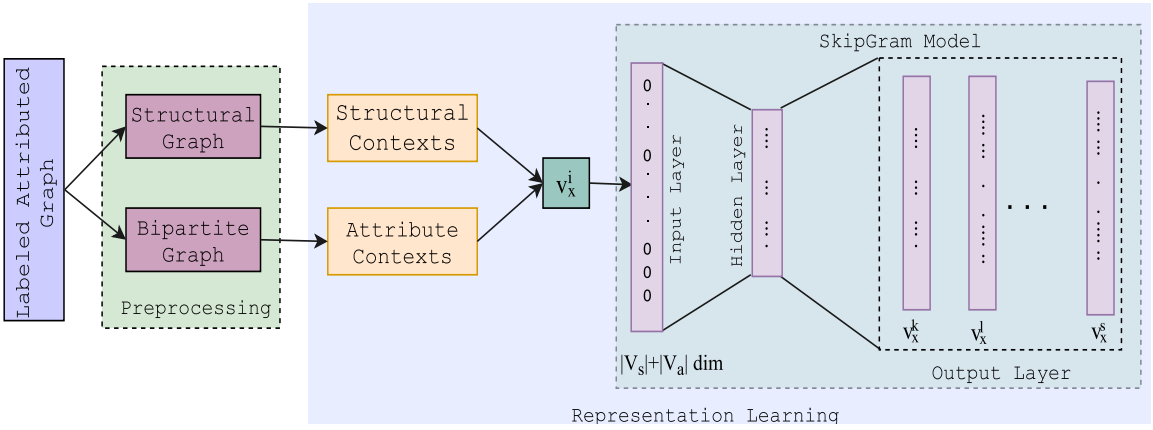

Figure 5.4: The Architecture of GAT2VEC.

Algorithm 1: The GAT2VEC Algorithm

Input : An Attributed Graph $G = (V, E, A)$
Output : $\Phi : V \rightarrow \mathbb{R}^d$, $d \ll |V|$
Parameters: walks per node γ_s, γ_a;
length of walks λ_s, λ_a;
context window size c;
embedding size d

1 Obtain G_s and G_a from G
2 **for** $v \in V_s \cup V_a$ **do**
3 | initialize $\Phi(v)$ // initializing the vectors
4 **end**
5 **for** $u \in V_s$ **do**
6 | $R_s(u) = \text{RandomWalks}(G_s, u, \gamma_s, \lambda_s)$
7 **end**
8 **for** $v \in V_a$ **do**
9 | $W_a(j) = \text{RandomWalks}(G_a, v, \gamma_a, \lambda_a)$
10 **end**
11 $R = \oplus_{u \in V_s} R_s(u)$
12 $W = \oplus_{v \in V_a} W_a(v)$
13 $\Phi = SkipGram(R, W, c)$ // as per equation. 5.1
14 return Φ

The architecture of GAT2VEC is shown in Figure 5.4. The input is given by structural and attribute contexts obtained from the respective graphs (line 11, 12 of Alg. 1). We use the SKIPGRAM model to jointly learn an embedding based on these two contexts. From each context, structural or attribute, a vertex $V_x \in V_s|V_a$ is selected and is input to SKIPGRAM. The input vertex is one-hot encoded vector $\{0, 1\}^{|V_s \cup V_a|}$. The input vertex V_x is the target vertex and the output layer produces $2c$ multinomial distributions of associated context vertices to the given input vertex. c is the context size i.e, the number of predicted vertices before or after the target vertex. Likewise, the output vertices can belong either to structural vertices or to attribute vertices, or to both, depending on their co-occurrence

in random walks.

Given a target vertex, the objective of GAT2VEC model is to maximize the probability of its structural and attribute contexts. Similar to previous studies [26, 28], we follow the assumption that given a target vertex, the probability of context vertices are independent of each other. Therefore, learning is described by the following objective function:

$$L = \sum_{r \in R} \sum_{i=1}^{|r|} \log p(r_{-c} : r_c | r_i) + \sum_{w \in W} \sum_{i=1}^{|w|} \log p(w_{-c} : w_c | w_i) \tag{5.1}$$

The equation 5.1 can be written as:

$$L = \sum_{r \in R} \sum_{i=1}^{|r|} \sum_{\substack{-c \leq j \leq c \\ j \neq i}} \log p(r_j | r_i) + \sum_{w \in W} \sum_{i=1}^{|w|} \sum_{\substack{-c \leq t \leq c \\ t \neq i}} \log p(w_t | w_i) \tag{5.2}$$

where $r_{-c} : r_c$ and $w_{-c} : w_c$ correspond to a sequence of vertices inside a contextual window of length $2c$ in random walks contained in corpus R and W, respectively.

The first term uses the structural contexts, while the second is for learning from attribute contexts. If $|V_a| = 0$, then the model will become DEEPWALK, i.e learns only from structure. The term $p(r_j|r_i)$ is the probability of the j-th vertex when r_i is the central vertex in the structural context r, while the term $p(w_t|w_i)$ is the probability of the t^{th} vertex when w_i is the central vertex in the attribute context w. These probabilities can be computed using the softmax function. The probability $p(r_j|r_i)$ can be computed as:

$$p(r_j | r_i) = \frac{\exp\left(\varphi(r_j)^T \Phi(r_i)\right)}{\sum_{v_s \in V_s} \exp\left(\varphi(v_s)^T \Phi(r_i)\right)} \tag{5.3}$$

where $\varphi(\cdot)$, $\Phi(\cdot)$ are representations of a vertex when it is considered as a context vertex or a target vertex respectively. Similarly, we can compute $p(w_t|w_i)$ following Equation 5.3.

The softmax calculation is computationally expensive due to normalization over all vertices of a graph. Thus, we approximate it by using the hierarchical softmax function [14]. Following [63], we used Huffman coding to build binary trees for hierarchical softmax which has vertices as leaves. Therefore, in order to compute the probability, we just need to follow the path from the root to the leaf node of the tree. Thus, the probability of a leaf node r_j to appear in the structural context is:

$$p(r_j|r_i) = \prod_{h=1}^{\mathrm{d}} p(s_h|\Phi(r_i)), \tag{5.4}$$

where $d = \log |V_s|$ is the depth of the tree and s_h are the nodes in the path with $s_o =$ being the root and $s_d = r_j$. Furthermore, modeling $p(r_j|r_i)$ as a binary classifier reduces the computational complexity to the order of $O(\log(|V_s|)$. The same can be applied to compute the probability for vertices in attribute contexts. Given that we are computing the probabilities from two contexts, this leads to the overall computational complexity of $O\big(\log |V_s| + \log |V_a|\big)$.

5.2.4 gat2vec-wl

In our work, we can also exploit the labels for precise learning of an embedding in a semi-supervised manner. The idea behind this approach is that the vertices sharing labels are similar and thus should be encoded close to each other in the embedding. We propose GAT2VEC-WL to incorporate labels associated with content vertices as attributes of vertices. The labels will be defined as an attribute vertex in the bipartite graph defined in Section 5.2.1. The labels will be helpful in generating the contexts in which the vertices sharing labels will appear together.

Since real-world networks are partially labeled, therefore, we randomly pick a percentage L_P of labeled content vertices and incorporate their labels

as attributes. After learning an embedding, these selected vertices are used in subsequent machine learning tasks. For example, for classification, we train our classifier on these selected vertices and predict labels for the rest.

5.3 Experiments

In this section, we provide an overview of widely used datasets and discuss the details of the experimental setup to compare the performance of our proposed approach, GAT2VEC, against a collection of state-of-the-art approaches.

We will validate the representation learned so far to perform two important tasks, namely vertex classification and link prediction.

5.3.1 Datasets

We use three real-world datasets: the social network BLOGCATALOG [64] and the citation networks DBLP [65] and CITESEER [66]. Table 5.1 summarizes their statistics, including information about the associated structural and attribute graphs.

DBLP and CITESEER are widely used for experimentation [48]. DBLP contains bibliographic data in computer science, extracted from a selected list of 34 conferences from 4 research areas. CITESEER is a multi-disciplinary dataset consisting of papers from 10 research fields.

In the DBLP and CITESEER datasets, the title of the paper constitutes the vertex content. We pre-process these titles to remove stop words. From each dataset, we selected the words occurring at least th times as vertex attributes. At $th = 3$, DBLP, CITESEER and BLOGCATALOG have 8618, 3986, and 5413 attributes, respectively.

BLOGCATALOG [67] is a social network of bloggers. The labels represent the interests of bloggers and each blogger may be associated to multiple

Dataset	$\|V\|$	$\|E\|$	# Lbls	#Lbl Vertices	V_s	V_a	$\|\mathbb{A}\|$	$\|E_a\|$
DBLP	60,744	52,890	4	60,744	17,725	60,720	8618	356,230
CiteSeer	38,996	77,218	10	10,310	36,227	10,107	3986	59,477
BlogCatalog	70,004	1,409,112	60	70,004	55,771	57,709	5413	269,363

Table 5.1: Dataset Statistics

labels. The attributes are keywords generated from users blog.

5.3.2 Baseline Methods

We compare GAT2VEC against state-of-the-art NRL algorithms for learning graph embeddings: two structure-based methods (Node2Vec and DeepWalk), one content-based (Doc2Vec), and two methods using both structure and content (TriDNR and TADW). We also consider variants of TriDNR and TADW. All the results in the experiments are obtained using the code released by the authors.

- Node2Vec [28] uses biased random walks to generate the vertex sequences based on in-out parameters p, q. For each of our datasets, we learned these hyper-parameters as described in the original paper.

- DeepWalk [26] is an approach based on random uniform walks to learn a d-dimensional feature representations of vertices in a network using only structure information.

- Doc2Vec [63] is an unsupervised neural network model to learn a representation for variable length texts such as sentences, paragraphs or documents. The model learns both word vectors and document vectors. We fed the text associated with each vertex to the model and obtained a representation for each vertex.

- TriDNR [48] is a learning model which uses three sources of information: network structure, vertex content, and label information to learn the representation of vertices. TriDNR uses two models: DeepWalk to learn the representation from structure, and Doc2Vec to capture the context related to node content and label information. The final representation is a linear combination of outputs of these two models.
- TriDNR noLbl is an unsupervised variant of TriDNR which learns a representation without using vertex labels.
- TADW [44] learns a d-dimensional representation of vertices by factorizing a text-associated matrix. Before factorizing the matrix, the features are reduced using `SVD`.
- TADW noSVD is a modified version of TADW in which we fetched raw feature vectors of vertices without performing `SVD` on them to learn a representation of a network.
- gat2vec-wl is a semi-supervised variant of gat2vec to learn a representation using labels as attributes.
- gat2vec-bip is another version of gat2vec which learns a representation on the bipartite graph only. Thus, no structural information is used for learning.

5.3.3 Experimental Setup & Parameter Settings

We perform multi-class classification on dblp and CiteSeer, and multi-label classification on BlogCatalog using the respective learned representation.

We set the parameters of gat2vec as follow: number of walks, $\gamma_s = 10$ and $\gamma_a = 10$; walk length, $\lambda_s = 80$ and $\lambda_a = 80$. The representation size is

$d = 128$; window size is $c = 5$, the threshold th is equal to 3. For fairness of comparison, the parameters that are in common between GAT2VEC and the other methods are set with the same value, while the rest of the parameters are set to their optimal default values as reported in the respective papers.

5.3.4 Vertex Classification

For classification, we used one-vs-rest logistic regression classifier LIBLINEAR [68] with default parameters for training the data, and then predict the unlabeled vertices. We randomly selected a sample of size $T_R \in \{10\%, 30\%, 50\%\}$ of vertices as training set to train the classifier and used the rest as a test set.

In case of GAT2VEC-WL, we randomly selected a percentage of labeled vertices, $L_P \in \{10\%, 30\%, 50\%\}$ and incorporated their values as attributes for learning a representation. Then these labeled vertices are used in training the classifier and predicting the labels of the remaining vertices.

The metrics used to compare our approach against the baselines are the widely used MICRO-F1 and MACRO-F1 scores. For fairness in comparison, we used the same training set of GAT2VEC across all baselines for training the classifier. For each training ratio, we repeat the process 10 times and report the average scores.

Results and Discussion

Tables 5.2–5.4 show the MICRO-F1 and MACRO-F1 results on the DBLP, CITESEER and BLOGCATALOG datasets. The results are consistent with the results reported in the baselines. GAT2VEC outperforms all baseline methods in all three datasets, even when a small number ($T_R = 10\%$) of vertices are used for training. This makes GAT2VEC suitable in real-world, sparse-labeled datasets.

Method	Micro-F1			Macro-F1		
	10%	*30%*	*50%*	*10%*	*30%*	*50%*
DeepWalk	51.8	52.5	52.7	41.0	42.1	42.3
Node2Vec	48.9	49.3	50.0	44.7	45.4	45.7
Doc2Vec	74.2	76.1	78.0	67.2	70.0	71.2
gat2vec-bip	76.5	77.4	77.6	70.1	70.8	71.1
TADW	68.4	68.8	69.0	62.9	63.1	63.5
TADW noSVD	60.8	61.0	61.1	56.1	56.5	56.5
TriDNR noLbl	74.3	74.8	75.1	67.0	67.7	73.1
gat2vec	**79.0**	**79.4**	**79.5**	**72.6**	**73.1**	**73.4**

Table 5.2: Multi-class Classification on dblp

Method	Micro-F1			Macro-F1		
	10%	*30%*	*50%*	*10%*	*30%*	*50%*
DeepWalk	44.0	47.8	49.0	35.1	37.9	39.0
Node2Vec	63.7	66.5	67.3	54.6	56.3	57.0
Doc2Vec	60.9	70.0	**72.8**	55.5	66.4	**68.8**
gat2vec-bip	65.5	70.2	71.4	61.4	66.5	67.8
TADW	49.0	51.2	51.8	44.0	45.8	46.4
TADW noSVD	32.5	35.2	36.0	28.4	30.5	31.4
TriDNR noLbl	63.3	67.6	69.8	57.8	62.0	64.5
gat2vec	**66.3**	**70.7**	72.0	**62.2**	**67.1**	68.5

Table 5.3: Multi-class Classification on CiteSeer

The results validate our approach of generating structural and attribute contexts and then subsequently learn the representation from both of them. In fact, gat2vec beats the state-of-the-art, attribute-based method TriDNR in all three datasets. This implies that properly modeling attribute information increases the preciseness of embeddings.

Furthermore, gat2vec outperforms TADW in all datasets. This is due to the structural and attribute sparsity. The impact of attribute sparsity

Method	Micro-F1			Macro-F1		
	10%	*30%*	*50%*	*10%*	*30%*	*50%*
DeepWalk	29.8	31.8	32.2	16.2	17.8	18.6
Node2Vec	31.0	32.5	32.8	17.0	18.6	18.8
Doc2Vec	40.3	41.4	41.9	23.4	25.8	26.6
gat2vec-bip	47.3	49.2	49.6	34.0	36.6	37.0
TADW	38.0	38.8	39.0	23.4	24.3	24.6
TADW noSVD	24.8	25.4	25.9	8.1	8.2	8.5
TriDNR noLbl	34.2	36.3	36.8	21.0	21.5	22.2
gat2vec	**49.5**	**51.2**	**51.6**	**36.3**	**38.8**	**39.3**

Table 5.4: Multi-label Classification on BlogCatalog

can be ascertained from results of CiteSeer dataset which is much more sparse than dblp, as only 10,310 out of 38,996 vertices have titles associated with them. This is also the reason of deviation from reported results in TriDNR which learns a representation using only 10,310 vertices. The other reason for this poor performance is due to using an approximation of DeepWalk. The dismal performance of TADW noSVD shows high dependence of the TADW approach on the costly `SVD` method; without `SVD`, the sparsity issue is aggravated as shown in Table 5.3 for the CiteSeer dataset.

gat2vec-bip outperforms the content-based method Doc2Vec in BlogCatalog dataset, but has almost similar performance in case of CiteSeer and dblp datasets. This highlights the appropriate modeling of attribute information even in cases when attribute information is non-cohesive and semantically unrelated such as in BlogCatalog dataset.

In the case of CiteSeer and dblp datasets, Doc2Vec uses only titles of papers which has rich information as compared to the structure. Thus, it performs better than DeepWalk but has comparable performance with Node2Vec as it exploits the structural information much efficiently than

DeepWalk. Furthermore, in BlogCatalog dataset, Doc2Vec underperforms as the contents do not contain rich information. In this case, our modeling proves to be better as shown by results of gat2vec-bip.

Structure-based methods such as DeepWalk and Node2Vec perform poorly, justifying the use of information from multiple sources to learn embeddings.

Table 5.5 shows the Macro-F1 of the three datasets when the labels are incorporated in learning the embedding. We have included just TriDNR as it is the only baseline which uses labels and attributes in learning. The results of gat2vec prove our claim that including the labels increases the quality of embedding in classification task. Furthermore, it also implies that our proposed approach of incorporating attributes is sufficient to generate the proper contexts. gat2vec has superior performances with respect to TriDNR, which in turn implies that performs better than the rest of the baselines.

Dataset	**Tr**	TriDNR	gat2vec-wl
dblp	10%	69.5	**75.2**
	20%	72.0	**81.4**
	30%	72.2	**86.3**
CiteSeer	10%	64.0	**79.2**
	20%	71.1	**77.0**
	30%	73.8	**83.0**
BlogCatalog	10%	21.0	**39.7**
	20%	23.8	**53.0**
	30%	24.9	**63.8**

Table 5.5: Macro-F1 score of classification (using labels)

5.3.5 Link Prediction

One important network analysis task is link prediction, and in the following we present the setting and comparison of GAT2VEC and the baselines for this task. We follow a similar strategy as [27, 28] and sample 15% of the observed or true edges ($T_E \subset E$) and remove them from the graph while ensuring the residual network is connected. We also sample 15% false edges ($F_E \bigcap E = \emptyset$). Next, each network representation algorithm is trained using the residual network. Once the training is complete, we get an embedding for each node. Then we predict the probability of an edge (u, v) being a true edge, for all edges $(u, v) \in T_E \bigcup F_E$ that we have just sampled, according to the following equation.

$$p(u, v) = \frac{1}{1 + \exp^{-\Phi(u).\Phi(v)}} \tag{5.5}$$

We finally rank edges according to the probability p and consider the top-k ones in the ranking at different values of k and measure the prediction performance using precision at $k, P(k)$. Let R be set of edges ranked according to p and R_k be the top-k elements of R. Then, $P(k)$ computes the fraction of correctly predicted edges from the top-k predictions, and it is specified by

$$P(k) = \frac{|T_E \bigcap R_k|}{|R_k|} \tag{5.6}$$

	DEEPWALK	NODE2VEC	TADW	TRIDNR	GAT2VEC
P(1000)	0.93	0.92	**1.0**	0.95	0.99
P(5000)	0.67	0.73	0.80	0.70	**0.93**
P(10000)	0.54	0.59	0.54	0.52	**0.68**
P(15000)	0.43	0.45	0.41	0.42	**0.48**
P(20000)	0.34	0.34	0.33	0.33	**0.37**
P(25000)	0.27	0.27	0.37	**0.38**	0.30

Table 5.6: $P(k)$ for Link Prediction on DBLP

	DeepWalk	Node2Vec	TADW	TriDNR	gat2vec
P(1000)	0.85	**0.92**	0.20	0.81	0.74
P(5000)	0.70	**0.71**	0.17	0.89	0.64
P(10000)	**0.66**	0.65	0.18	0.51	0.63
P(15000)	**0.63**	**0.63**	0.19	0.49	**0.63**
P(20000)	0.62	0.61	0.20	0.48	**0.63**
P(25000)	0.61	0.60	0.21	0.47	**0.62**

Table 5.7: $P(k)$ for Link Prediction on BlogCatalog

Results and Discussion

The results for link prediction is given in Table 5.6–5.7. One can notice that for the dblp dataset (Table 5.6), where the structural network is very sparse, algorithms which are based only on the structural knowledge produce poor results. Our algorithm, which alleviate the structural sparsity by considering connections with attributes, significantly outperforms the baselines including those that integrate attribute and label information. Our finding also shows that when the structural network is sufficiently dense (BlogCatalog), structure based algorithms could be sufficient in link prediction as shown in Table 5.7.

5.3.6 Qualitative Analysis

To evaluate qualitatively the learned representations from our proposed approach, we performed Nearest Neighbor Search and visualization of the learned embeddings.

Case Study

We carried out a case study on the dblp dataset, by selecting a query paper and then obtaining the three nearest neighbors with respect to its representation, using the Cosine Distance metric. We generated an embed-

Algorithm	*Query*: Group Formation in Large Social Networks: Membership, Growth, and Evolution	Cites	# Cit.
gat2vec	1. Microscopic Evolution of Social Networks	Yes	4
	2. Structure and Evolution of On-line Social Networks	No	7
	3. Maximizing the Spread of Influence through a Social Network	Yes	14
TriDNR	1. Searching for Rising Stars in Bibliography Networks	Yes	0
	2. A Framework for Community Identification in Dynamic Social Networks	Yes	9
	3. A Framework for Analysis of Dynamic Social Networks	Yes	2
TADW	1. Tutorial summary: Large social and information networks: opportunities for ML	No	0
	2. Exploring and Visualizing Academic Social Networks	No	0
	3. Seeing Sounds: Exploring Musical Social Networks	No	0
Node2Vec	1. Characterizing and Predicting Community Members from Evolutionary and Heterogeneous Networks	Yes	0
	2. Searching for Rising Stars in Bibliography Networks	Yes	0
	3. Constant-factor Approximation Algorithms for Identifying Dynamic Communities	Yes	0

Table 5.8: Nearest Neighbor Top 3 Results

ding through GAT2VEC-WL, with 10% vertices labeled. The query paper is *"Group Formation in Large Social Networks: Membership, Growth, and Evolution"*, which has the highest number of citations in DBLP (1666 on July 2017). The paper studies the effect of network structure on the evolution of communities in social networks. Therefore, the results of the near-

est neighbors should have attribute contexts related with social networks, communities, and evolution. In addition, the resulting papers should also preserve its structural contexts. Table 5.8 lists the query results on different embeddings, along with citation relationship, and common citations between the query and the result.

The results returned by GAT2VEC are relevant in attribute contexts to the query. The first and third results are cited by the query, which shows that GAT2VEC preserves the direct proximity. Indirect proximities are preserved as well, as it evidenced by the second query result which is not cited by the query but has 7 common cited papers. Therefore, our proposed method GAT2VEC generates accurate structural and attribute contexts which eventually help to learn precise embeddings.

Apart from the first one, the results from TRIDNR are quite related with the query as they are either directly cited or have common citations with query paper. The results from TADW have some relevance with the query. But structurally, there is neither citation relationship nor any common cited papers with the query paper. The possible reason for such poor results could be due to matrix approximation for obtaining structural contexts. NODE2VEC results show what happens when ignoring the attribute information, nonetheless NODE2VEC exploits the structural information as the results and query have a direct relationship.

The above qualitative analysis support the claim that using multiple sources of information aids in learning the precise embeddings.

Network Visualization

The learned embedding can be projected in a two dimensional plane to visualize the co-relationship between the nodes. We use the learned embedding from TRIDNR, TADW, and NODE2VEC on the CITESEER dataset for comparison. Papers which cite less than five papers are filtered out. Since

these embeddings are in a higher dimensional space, we use t-SNE [69] to reduce these to 2-D space. The visualizations of different approaches are given in Figure 5.5.

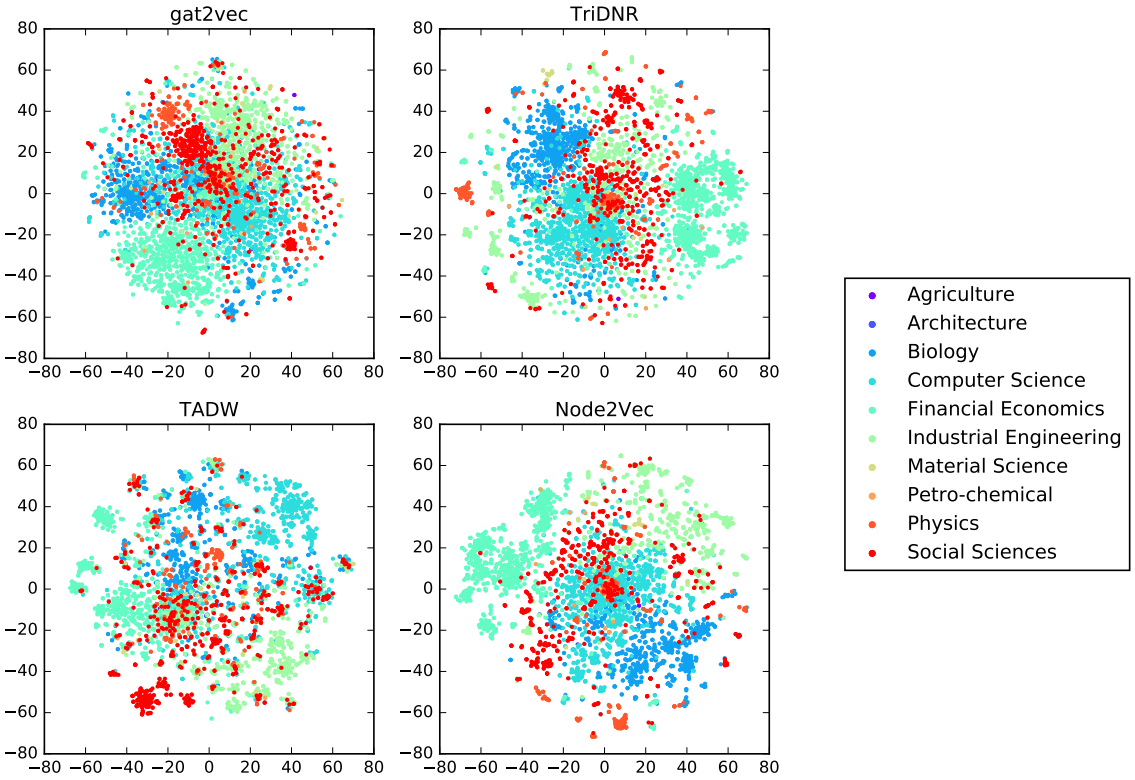

Figure 5.5: 2-D t-SNE Projection of CiteSeer Dataset

The visualization using TADW is not very meaningful; in fact, the papers belonging to *Social Sciences, Biology* are not clustered. A small dispersion of *Social Sciences* papers is acceptable as they are cross-cited across different fields such as *Computer Science*. Unfortunately, no proper cohesive group is formed here. The problem is related to the use of the approximation approach for random walks. The results from Node2Vec are much better, as it depicts some communities properly (*Financial Economics, Computer Science*). This is because Node2Vec performs biased walks which are effective in capturing the structural equivalences.

TriDNR performs quite well as some clearly defined clusters can be

seen compared to other two approaches. It still does not learn meaningful embeddings as papers from *Social Sciences* and *Industrial Engineering* are not clustered but are dispersed. Our proposed approach GAT2VEC performs better than all given methods. The clusters are well-formed and depict the overlapping properly. The papers from *Social Sciences* can be seen forming a well-defined cluster in contrast to other approaches. In addition to it, this cluster is close to *Computer Science* and *Financial Economics* clusters, which is plausible as there are cross citations between these fields. Thus, our proposed approach learns an embedding which preserves the structural and attribute equivalence in the underlying graph.

5.3.7 Parameter Sensitivity

GAT2VEC requires various parameters to create G_s and G_a, and learn an embedding. The sensitivity of the choice of parameters on structural networks have been investigated in previous works, such as [26, 28]. We investigate the parameter sensitivity of γ_a, λ_a, and *th* on attributed graph. Furthermore, we analyze the joint parameter sensitivity of number of walks (γ_s, γ_a), and walk length (λ_s, λ_a) on G_s and G_a. In these experiments, we evaluate on MICRO-F1 under the same evaluation process given in Section 5.3.4. All other parameters are set to default values unless mentioned.

Impact of γ_a & λ_a

Figures 5.6(a,b) shows the effects of varying the number of walks per vertex and the walk length on bipartite attribute graphs. We empirically observe that these parameters have direct proportionality relationship with the sparsity. In a very dense graph, a small number of short walks are adequate to explore the neighborhood to capture the contexts. DBLP is highly dense, while CITESEER is very sparse, as shown in Figure 5.8. Therefore, for DBLP, our framework is able to generate precise representations at lower

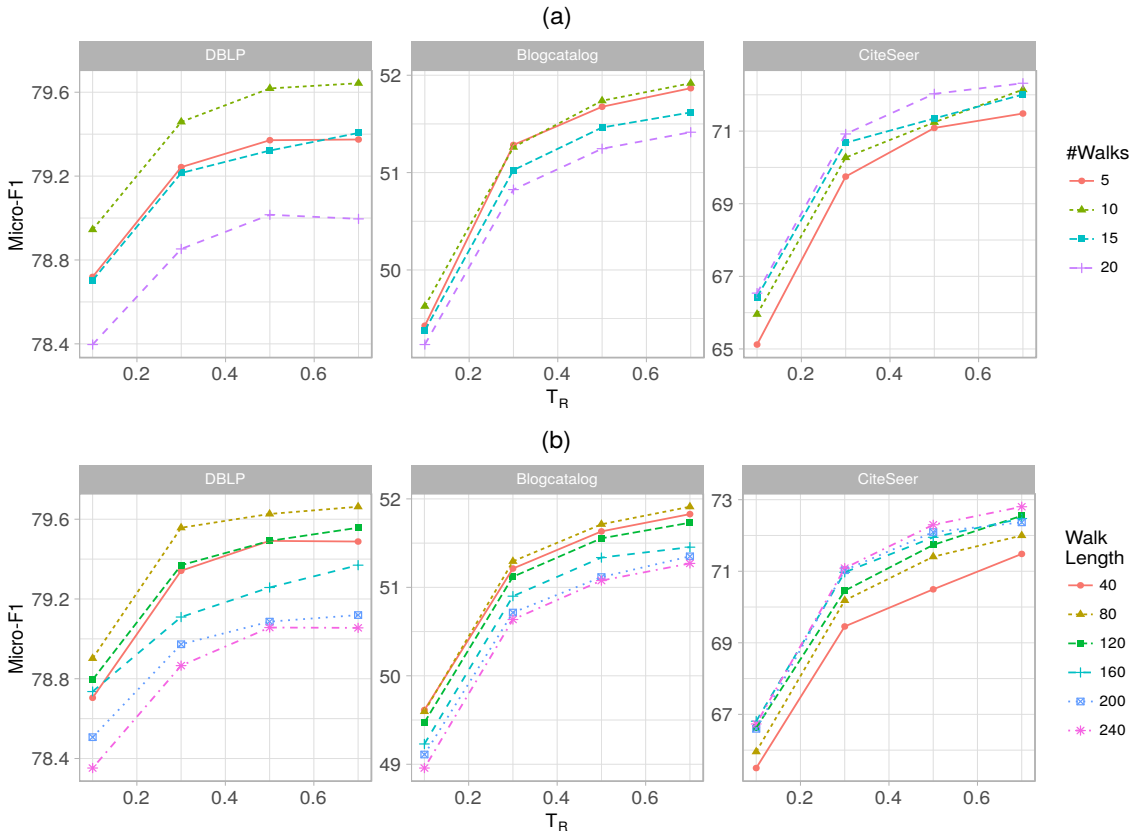

Figure 5.6: G_a Parameter Sensitivity on: (a) Number of Walks(γ_a), (b) Walk Length(λ_a)

values of γ_a and λ_a. Increasing the number of walks and the walk length has detrimental effect on the learning representation as it explores the large parts of the graph, which in turn generates contexts that are noisy i.e., include less significant vertices. In case of CITESEER, we see an opposite trend: precise representations are learned for higher values of γ_a and λ_a. This is motivated by the sparsity of the dataset, which is composed of 10 different disciplines, thus requiring a larger number of longer walks to explore the graph adequately and generate the appropriate contexts.

Impact of γ_s, γ_a, & λ_s, λ_a

Here we analyze the joint parameter sensitivity on the DBLP dataset. In Figure 5.7(a) we increase the number of walks (γ_s, γ_a) jointly on both G_s

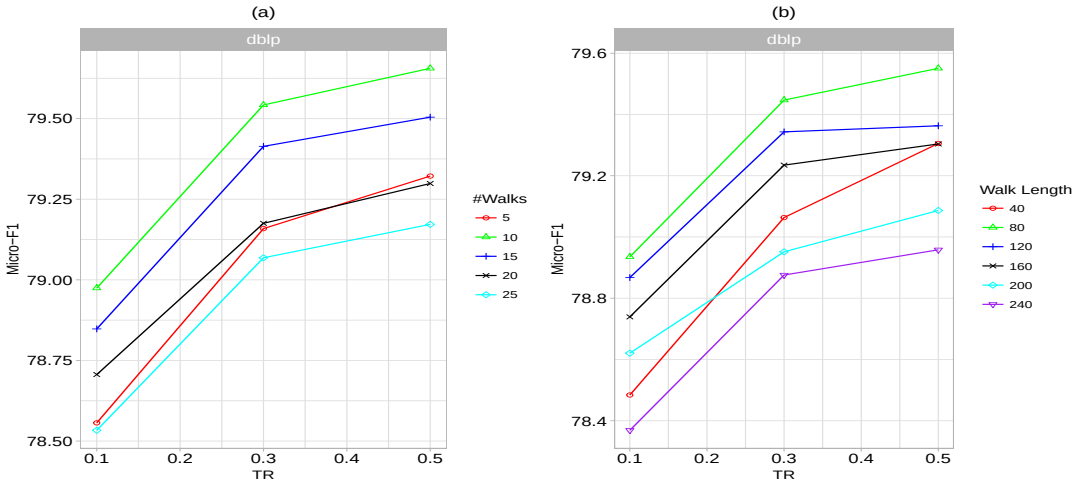

Figure 5.7: G_a Joint Parameter Sensitivity on: (a) Number of Walks(γ_a), (b) Walk Length(λ_a)

and G_a, while keeping walk length to 10. In second case 5.7(b), walk lengths are jointly increased, while the number of walks is kept constant to 10. As discussed in previous case, increasing the number of walks or walk length gathers more contextual information, thus, learning more precise representations. But, an excessive number of walks and large walk lengths are detrimental as it generates noisy contexts which lead to poor representation, as shown in case $\gamma_s = \gamma_a = 25$ (Figure 5.7a), and $\lambda_s = \lambda_a = 240$ (Figure 5.7b).

Figure 5.8: Sparsity of Attributed Graph G_a

Impact of *th*

We have shown the usefulness of attributes (Table 5.2–5.4) for learning high quality embeddings. We further stress upon the importance of attributes by varying the parameter *th*. It is pertinent to mention that increasing the value of *th* implies a decrease in the number of attributes. From Figure 5.9, it is evident that as we decrease the number of attributes, the accuracy of embeddings decreases. At $th = 1$, the performance is lower due to the noise introduced by including all possible words as attributes.

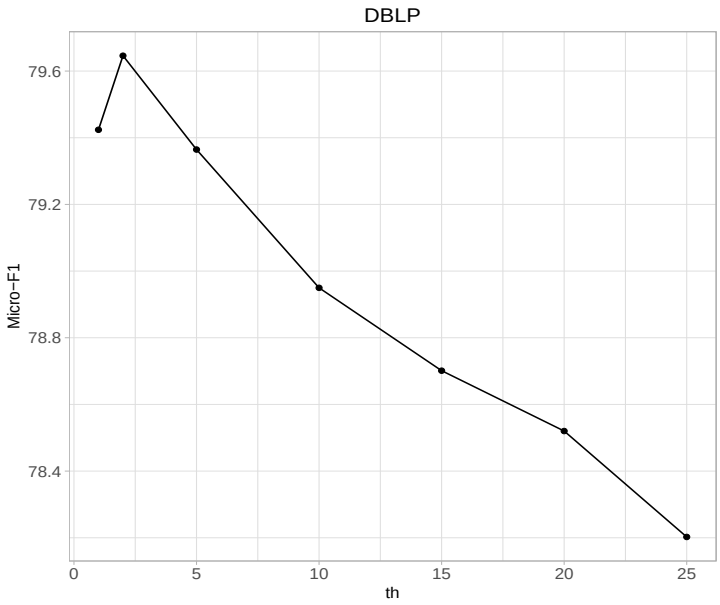

Figure 5.9: Effect of parameter-*th*

Chapter 6

A Simple Approach to Learn Representation on Attributed Graphs

Nodes in an attributed graph tend to have homophilic relationships, i.e. tend to connect to nodes that are similar to themselves in terms of attributes. By exploiting attribute information and learning from both the network structure and the attributes at the same time, better representations can be obtained. But this comes at a cost, due to the increase in the computational complexity needed to model the various relationships, structural and attribute, between nodes. Moreover, for some graphs the number of attributes may be very large (in order of millions), that increases the overall model complexity.

Similar to network structure, attributes are also highly non-linear and sparse [27, 33]. Various approaches have been proposed which handle the network structure and attribute nonlinearity, and sparsity [49, 50]. DANE [49] handles them by employing a deep neural network model on both the network structure and attributes to handle non-linearity and sparsity. This method preserves semantic proximities between nodes by sampling nodes that have high similarity in attributes and optimizing their represen-

tation together with the first- and higher-order proximities in the network structure. Unfortunately, the cost of this sampling procedure is quadratic in the number of nodes, thus not scalable to large graphs. Another approach called Attributed Network Representation Learning (ANRL) [50] uses a neighbor enhancement autoencoder which optimizes the learning on attributes and the neighborhood context. The neighborhood context is generated through random walks whose complexity is too high for large graphs, both in terms of time and space.

Therefore, handling attributes in conjunction with structure is more challenging as it involves capturing non-linearity and sparsity, while at the same time preserving the structural properties of the graph.

To address these problems, this chapter proposes SAGE2VEC (Simple Attributed Graph Embedding), a model that preserves the second-order proximities in the structure, and also handles the network structure and attribute non-linearity, and sparsity. The motivation behind this work is to build a simple model in terms of number of parameters and to show that a very simple approach can achieve better results than complex state-of-the-art baselines in a large percentage of scenarios. We employ an autoencoder model with an *enhanced decoder* that takes only structural information as input but optimizes on both structure and attribute information. By using attribute information for optimization reduces the overall model complexity and thus, the model can be scaled to larger graphs.

The rest of the chapter is organized as follows: Section 6.1 provides the formal definition of the problem. In Section 6.2, we describe our proposed model and the experimental setup, which is later evaluated on several datasets in Section 6.3.

6.1 Problem Definition

Problem Given an attributed graph G, we aim at learning a low-dimensional network representation $\Phi : V \rightarrow \mathbb{R}^d$, where $d \ll |V|$ is the dimension of the learned representation, such that the mapping function Φ preserves the structural proximity and attribute information.

The quality of this mapping will be evaluated against several tasks such as node classification on various real-world datasets, as described in Section 6.3.

6.2 Model

We describe now SAGE2VEC, a novel model that learns an embedding using both the network structure and the attributes. It is essential that the learning occurs on both elements at the same time, so that the two modalities complement each other, and not as a combination of two learned models. In this way, the learned representation preserves the structural proximity and encodes the attribute information in it.

We propose an autoencoder with enhanced decoder model which jointly learns a representation using the network structure and the attributes. Our model preserves the second-order proximity, captures the non-linearity and addresses the sparsity of both the network structure and the node attributes.

Enhanced Autoencoder Our enhanced autoencoder model considers the non-linearity of network attributes in conjunction with the network structure and jointly learns a representation from these two different modalities. Its architecture is shown in Figure 6.1. The encoder part is similar to Equation 3.8. The input to our model is only network information. For a given vertex i, our model takes its second-order proximity $\mathbf{M}_i$ as input, and

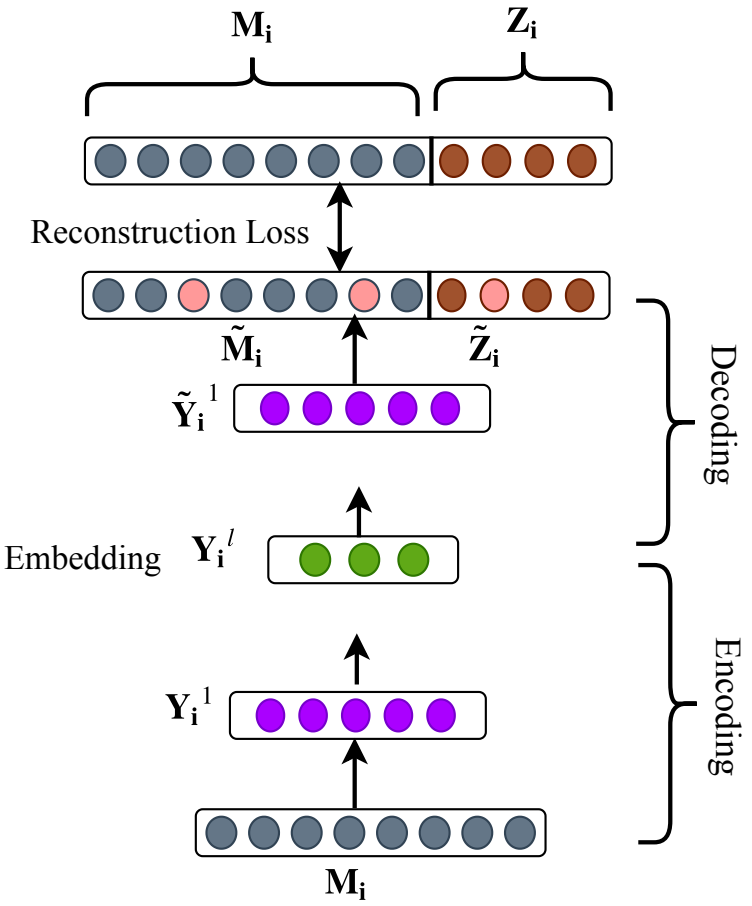

Figure 6.1: The architecture of our proposed SAGE2VEC model

aims to reconstruct the neighbors $\tilde{\mathbf{M}}_i$ along with its respective attributes $\tilde{\mathbf{Z}}_i$ by incorporating prior knowledge of attributes. To accommodate the reconstruction of attributes, Equation 3.7 is modified as follows:

$$\mathcal{L}_{ea} = \sum_{i=1}^{n} \|(\mathbf{M}_i - \tilde{\mathbf{M}}_i)\|_2^2 + \sum_{i=1}^{n} \|(\mathbf{Z}_i - \tilde{\mathbf{Z}}_i)\|_2^2 \tag{6.1}$$

Since both the network and the attributes are sparse, the autoencoder tends to reconstruct zeros. Similar to Wang et al., we impose a higher penalty to the reconstruction error of the non-zero elements than zero elements [27]. To incorporate the penalties, Equation 6.1 is revised as:

$$\begin{aligned}\mathcal{L}_{ea} &= \sum_{i=1}^{n} \|(\mathbf{M}_i - \tilde{\mathbf{M}}_i) \odot \mathbf{C}_i^s\|_2^2 + \sum_{i=1}^{n} \|(\mathbf{Z}_i - \tilde{\mathbf{Z}}_i) \odot \mathbf{C}_i^a\|_2^2 \\ &= \|(\mathbf{M} - \tilde{\mathbf{M}}) \odot \mathbf{C}^s\|_F^2 + \|(\mathbf{Z} - \tilde{\mathbf{Z}}) \odot \mathbf{C}^a\|_F^2\end{aligned} \tag{6.2}$$

where $\odot$ is the Hadamard product, $\mathbf{C}_i^s = [\mathbf{C}_{i1}^s, \mathbf{C}_{i2}^s, \cdots, \mathbf{C}_{in}^s]$ and $\mathbf{C}_i^a = [\mathbf{C}_{i1}^a, \mathbf{C}_{i2}^a, \cdots, \mathbf{C}_{iz}^a]$ are the penalties for network and attribute zero reconstructions, respectively. The penalty for zero reconstruction when the dealing with the network structure is given as:

$$C_{ij}^s = \begin{cases} 1, & \text{if } \mathbf{M}_{ij} = 0 \\ \beta, & \text{otherwise} \end{cases} \tag{6.3}$$

where $\beta > 1$. Likewise, the penalties for zero reconstructions in attributes are similar to Equation 6.3.

To prevent overfitting, we add a l_2 regularizer. Thus, the overall objective is to minimize the following loss function:

$$\begin{aligned}\mathcal{L}_o &= \mathcal{L}_{ea} + \gamma \mathcal{L}_{reg} \\ &= \|(\mathbf{M} - \tilde{\mathbf{M}}) \odot \mathbf{C}^s\|_F^2 + \|(\mathbf{Z} - \tilde{\mathbf{Z}}) \odot \mathbf{C}^a\|_F^2 \\ &\quad + \gamma \sum_{l=1}^{L} (\|\mathbf{W}^{(l)}\|_F^2 + \|\tilde{\mathbf{W}}^{(l)}\|_F^2)\end{aligned} \tag{6.4}$$

where $\mathbf{W}^{(l)}$ and $\tilde{\mathbf{W}}^{(l)}$ are the weight matrices at l-th layer encoder and decoder, respectively, and γ is the regularization coefficient.

We use stochastic gradient descent to minimize the objective function $\mathcal{L}_o$ by using the back-propagation algorithm on the model parameters $\theta = \{\mathbf{W}^{(i)}, \tilde{\mathbf{W}}^{(i)}, \mathbf{b}^{(i)}\}$.

6.3 Experiments

In this section, we describe the datasets and machine learning tasks used to evaluate our proposed approach against the state-of-the-art baselines.

6.3.1 Datasets

We conduct the experiment on four benchmark datasets and the statistics are given in Table 6.1. CiteSeer, cora and pubmed[1] are citation networks. cora contains papers from machine learning, grouped in seven categories. CiteSeer contains papers from six categories corresponding to computer science fields. In both CiteSeer and cora, the attributes are 0/1 valued vectors for each node. The pubmed dataset is a citation network related to diabetes. wiki is a web graph of hyper-links [49]. The attribute vector of both pubmed and wiki for each vertex is described through a TF/IDF word vector.

reddit[2] graph is constructed from the reddit posts where a node is a post, and two posts are connected if the same user comments on both posts. The attributes are a concatenation of average word vectors of post's title and post's comment; post's score and the number of comments on the post [2]. The labels of nodes are the community memberships.

Datasets	$\|V\|$	$\|E\|$	$\|\mathbb{A}\|$	#Labels
CiteSeer	3,312	4,660	3,703	6
cora	2,708	5,278	1,433	7
pubmed	19,717	44,338	500	3
wiki	2,045	12,761	4,973	17
reddit	232,965	11,606,919	602	41

Table 6.1: Dataset Statistics

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

[2] http://snap.stanford.edu/graphsage/#datasets

6.3.2 Baselines

We evaluate our proposed approach SAGE2VEC with the several state-of-the-art baseline methods. DEEPWALK and NODE2VEC use only structure of network, while the rest use both the structure and the attributes to learn an embedding.

- DEEPWALK [26] employs short random walks to generate a corpus of vertex sequences, and then uses SKIPGRAM to learn a representation.
- NODE2VEC [28] uses biased short random walks to explore the diverse neighborhood to interpolate between *breadth first* and *depth first* sampling.
- VGAE [52] is based on *variational graph encoders* using graph convolutional network (GCN) and learns an embedding by using both structural and attribute information of vertices.
- GAE [52] is a non-probabilistic variant of VGAE.
- DANE [49] uses a deep network on both structural and attribute information, while maintaining the consistent and complimentary information between the two modalities of informations.
- ANRL [50] uses an attribute-aware SKIPGRAM model to incorporate attribute information and a neighbor-enhanced autoencoder to reconstruct the target neighbors. We use ANRL-WAN, the best-performing variant of ANRL introduced in the paper.
- CAN [51] uses a variational autoencoder to learn an embedding of vertices and attributes in the same semantic space. For evaluation, we use only the learned representations of nodes.

We used the code of the baselines released by the authors. The parameters used are the reported optimal parameters, otherwise we performed

a random search to obtain an optimal performance of the baseline for a given dataset. For our model, the number of layers and their sizes for four datasets is given in Table 6.2; the activation function is *tanh*, the *Adam* Optimizer [70] is employed, and the embedding size d is 128. Our model requires very less trainable parameters as can be seen in Table 6.2, thus can be trained quickly.

Dataset	#neurons in each layer
CiteSeer	3312-128-7015
cora	2708-128-4141
pubmed	19717-128-19217
wiki	2045-128-7018
reddit	232,965 - 1000 - 64 - 1000 - 233,567

Table 6.2: The Network Layer Structure for Enhanced Autoencoder

6.3.3 Vertex Classification

We evaluate the learned representations of our approach against baselines through multi-class classification. We use *one-vs-rest* as l_2 regularized logistic regression classifier. We randomly selected $T_r \in \{10\%, 30\%, 50\%\}$ vertices as training set and the rest as test set. We repeated the process 10 times for each T_r, and report the average Micro-F1 (Mi-F1) and Macro-F1 (Ma-F1).

The classification results of all datasets are shown in Table 6.3, 6.4, 6.5 and 6.6. The best performance is highlighted in bold. We have the following observations from the results:

- The results show that leveraging both the structure and attributes helps to learn a better representation, as expected from the previous works in this line.
- Our proposed approach achieves the best performance against all baselines in the cora, pubmed, wiki datasets and has competitive perfor-

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	49.48	42.84	53.85	46.64	55.36	48.60
Node2Vec	53.92	46.48	56.07	48.60	57.37	50.00
gae	63.68	55.26	64.51	56.0	65.0	56.40
vgae	61.25	53.90	62.65	54.78	63.12	52.01
dane	66.22	56.60	69.92	61.48	71.92	64.87
anrl	**72.24**	**63.60**	73.46	67.15	73.28	68.32
can	68.30	59.82	70.63	62.30	71.50	63.44
Sage2Vec	71.81	62.75	**73.74**	**67.30**	**74.89**	**69.42**

Table 6.3: Vertex Classification of CiteSeer

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	70.89	67.55	76.30	74.62	78.00	76.60
Node2Vec	74.00	70.66	78.15	77.17	79.32	78.60
gae	76.45	74.06	78.54	77.72	78.84	77.45
vgae	73.54	70.00	76.39	72.75	76.54	73.31
dane	77.80	73.75	81.97	80.45	82.97	81.42
anrl	73.55	69.51	76.55	73.85	78.24	74.50
can	78.68	75.55	82.22	79.48	82.57	80.77
Sage2Vec	**79.97**	**77.05**	**84.45**	**83.15**	**85.67**	**84.0**

Table 6.4: Vertex Classification of cora

mance with anrl for the CiteSeer dataset. Specifically, our model achieves an improvement around 11% in Macro-F1 for the wiki dataset against the second-best performing baseline (dane).

- The results show that a well-designed model can learn a better representation without involving complex and computationally intensive optimizations.

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	74.65	72.90	75.72	74.28	76.13	74.76
Node2Vec	80.27	78.77	80.90	79.52	81.20	79.85
GAE	82.03	81.34	82.27	81.63	82.38	81.78
VGAE	81.70	80.97	82.01	81.31	82.22	81.62
DANE	84.90	84.22	85.27	85.24	**86.76**	86.42
ANRL	84.55	84.69	85.11	85.25	86.32	**86.46**
CAN	78.52	78.20	79.05	78.65	79.38	78.97
Sage2Vec	**85.60**	**85.28**	**86.21**	**85.89**	86.55	86.22

Table 6.5: Vertex Classification of PUBMED

Method	10%		30%		50%	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
DeepWalk	54.78	35.86	61.25	41.80	62.54	44.25
Node2Vec	54.22	34.08	61.06	40.54	62.47	42.50
GAE	56.40	39.17	64.52	47.58	67.49	51.78
VGAE	60.02	40.21	63.94	44.94	64.93	46.46
DANE	65.13	44.25	73.85	54.0	75.95	56.82
ANRL	59.15	39.41	67.80	49.03	70.05	52.98
CAN	56.73	36.97	64.89	44.99	67.48	48.85
Sage2Vec	**68.79**	**48.15**	**77.01**	**57.80**	**79.42**	**63.15**

Table 6.6: Vertex Classification of WIKI

6.3.4 Link Prediction

The goal of link prediction is to predict edges between nodes. Following [28, 52], we generate a residual network by removing 10% of edges from the network, retaining all features and ensuring that the network remains connected. These removed edges are called true edges. We sample an equal percentage of negative edges (non-edges), called false edges. These true edges and false edges form a test set. We train the NRL models on the residual network. The models are evaluated based on their ability to correctly classify true edges and false edges. Similar to [50], we classify

Method	CiteSeer		cora		pubmed		wiki	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP
DeepWalk	0.83	0.87	0.86	0.89	0.84	0.86	0.90	0.92
Node2Vec	0.85	0.87	0.89	0.91	0.93	0.93	0.87	0.90
gae	0.90	0.89	0.92	0.92	0.94	0.94	0.91	0.93
vgae	0.93	0.94	0.92	0.93	0.92	0.92	0.92	0.94
dane	0.85	0.88	0.78	0.81	0.84	0.86	0.86	0.87
anrl	0.95	0.94	0.87	0.86	0.92	0.92	0.94	0.94
can	0.96	0.96	**0.92**	**0.93**	0.94	0.93	**0.96**	**0.97**
SAGE2Vec	**0.97**	**0.97**	**0.92**	**0.93**	**0.96**	**0.96**	0.95	0.96

Table 6.7: AUC and AP scores for Link Prediction

the true and false edges according to a ranking based on cosine similarity. We employ *area under ROC curve* (AUC) and *average precision* (AP) to evaluate the NRL models on the test set. We perform link prediction on all four datasets and the results are shown in Table 6.7. Once again, the results show the advantages of using structure and attribute information jointly. These two modalities complement each other and the network can show structure homophily, attribute homophily or both. Our model performs better in AUC and AP for the CiteSeer, pubmed datasets and is comparable for cora and wiki against all the baselines. In particular, our method achieves 2% gains in AUC and AP against the state-of-the-art baseline for the pubmed dataset.

6.3.5 Network Reconstruction

The purpose of network reconstruction is to test the preservation of the network structure in the learned embeddings. For each dataset, we learned an embedding using the baseline methods to predict the edges. Then we compute the probability of an edge $p(i, j)$ for each pair of node i, j given

as:

$$p(i,j) = \frac{1}{1 + exp^{-(\Phi(i).\Phi(j)^T)}} \tag{6.5}$$

where T is the transpose operation. We rank these edges from high to low probabilities. The edges from the original network serve as ground truth. We evaluate the performance of network reconstruction by correctly predicting the edges using *precision-at-K* ($P@K$) metric against the ground truth. K represents the number of predicted edges selected for evaluation. The results for all datasets are shown in Tables 6.8 - 6.11. The experiments show that our method performs better in $P@K$ in the CITESEER, CORA and WIKI datasets. In all these datasets, when K increases, $P@K$ is consistently higher than all baselines. In case of PUBMED, since the dataset is sparse, the NODE2VEC is able to explore properly the diverse neighborhood.

K	DeepWalk	Node2vec	GAE	VGAE	DANE	ANRL	CAN	SAGE2Vec
100	0.11	0.3	0.44	0.51	0.04	0.17	0.57	**0.58**
1000	0.14	0.5	0.3	0.39	0.03	0.08	0.46	**0.58**
3000	0.13	0.34	0.25	0.3	0.04	0.07	0.39	**0.53**
5000	0.14	0.31	0.21	0.26	0.19	0.06	0.37	**0.49**
10000	0.14	0.23	0.16	0.22	0.22	0.05	0.33	**0.42**

Table 6.8: Precision at K (P@K) for CITESEER dataset

K	DeepWalk	Node2vec	GAE	VGAE	DANE	ANRL	CAN	SAGE2Vec
100	0.1	0.52	0.49	**0.54**	0.0	0.08	0.52	0.43
1000	0.16	0.36	0.47	0.46	0.0	0.1	0.51	**0.55**
3000	0.15	0.18	0.37	0.39	0.05	0.07	0.45	**0.52**
5000	0.15	0.14	0.33	0.35	0.17	0.06	0.42	**0.47**
10000	0.14	0.12	0.27	0.28	0.19	0.05	0.36	**0.36**

Table 6.9: Precision at K (P@K) for CORA dataset

K	DeepWalk	Node2vec	GAE	VGAE	DANE	ANRL	CAN	SAGE2Vec
100	0.01	**0.36**	0.31	0.28	0.0	0.04	0.32	0.26
1000	0.03	**0.51**	0.24	0.18	0.0	0.03	0.18	0.14
3000	0.03	**0.43**	0.2	0.14	0.0	0.03	0.15	0.14
5000	0.03	**0.40**	0.18	0.13	0.0	0.03	0.14	0.13
10000	0.03	**0.33**	0.16	0.11	0.0	0.02	0.12	0.12

Table 6.10: Precision at K (P@K) for PUBMED dataset

K	DeepWalk	Node2vec	GAE	VGAE	DANE	ANRL	CAN	SAGE2Vec
100	0.43	0.23	0.06	**0.92**	0.5	0.08	0.51	0.87
1000	0.33	0.04	0.04	**0.88**	0.47	0.16	0.23	**0.88**
3000	0.32	0.09	0.04	0.78	0.55	0.19	0.2	**0.83**
5000	0.3	0.1	0.04	0.72	0.66	0.18	0.19	**0.77**
10000	0.28	0.1	0.05	0.58	0.58	0.19	0.15	**0.66**

Table 6.11: Precision at K (P@K) for WIKI dataset

6.3.6 Algorithmic and Scalability Analysis

In this section, we provide an analysis of the computational complexity of the baselines and our approach. The algorithmic computational complexity of the state-of-the-art models is provided in Table 6.12. In case of DANE, authors have used some simple sampling strategy but the algorithm is still dominated by a quadratic time complexity. Given that real-world large graphs have nodes and edges in the order of millions, the state-of-the-art methods require huge computational resources, thus limiting their usability. Our model SAGE2VEC has the smallest algorithmic computational complexity, and hence is scalable to larger graphs.

To perform the scalability analysis we used the larger graph available, REDDIT. We conducted the experiments on a 48-core machine based on the Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz processor, with 128GB of RAM. We report the asymptotic computational complexity of the algorithms and the wall-clock time in the Table 6.12. All the baselines ran out

Dataset	Algorithmic Computational Complexity	Training Time
GAE	$O(\|E\|)$	NA
DANE	$O(n^2)$	NA
ANRL	$O(n \log n)$	NA
CAN	$O(\|E\|)$	NA
SAGE2VEC	$O(n)$	≈10 hours

Table 6.12: Computational Complexity and Scalability Analysis on REDDIT dataset (NA:*out of memory*)

of memory; DANE and ANRL during the data preprocessing phase, while GAE and CAN during the model computation. We run SAGE2VEC for 5 epochs to show that the model works on larger graphs.

Dataset	End to End	Avg. Epoch Time
GAE	32.4	0.21
DANE	4345.5	0.45
ANRL	2730.0	**0.19**
CAN	43.0	0.35
SAGE2VEC	**27.5**	0.90

Table 6.13: Running Time Analysis on CITESEER dataset (in seconds)

In order to quantify the running time, we used the CITESEER dataset for demonstration. The end-to-end time is measured from the start of the program to its end, so as to highlight the preprocessing time in the baselines. Moreover, we also measured the execution time per epoch and we report the average over 10 epochs. Our model has the smallest end-to-end running time showing as it does not have high preprocessing costs. Though, the execution time per epoch is highest among baselines but it converges swiftly. The results are given in Table 6.13.

DANE and ANRL have the highest end-to-end running time, due to sampling and complex optimization costs. The results validate that a simple and well designed model can achieve better results and be more efficient

than the existing complex approaches.

Chapter 7

Heterogeneous Information Network Representation Learning

Existing NRL approaches have focused on embedding *homogeneous* networks, where only one type of nodes and edges exist. Real-world networks are by and large inherently heterogeneous - having distinct types of nodes and edges.

For example, consider a patent dataset, which consists of three different nodes: patent, inventor and assignee. A patent has semantically different relationships in the network. An inventor invents a patent, the patent is assigned to an assignee, and a patent cites other patents. Links connect both different and similar types of nodes. Due to different semantics of node relationships, the homogeneous representation learning methods cannot be applied because they cannot preserve the different semantics.

In this chapter, we propose HetNet2Vec, a novel approach to learn a representation of nodes in a heterogeneous information network (HIN). To preserve the semantic relationships, we perform relationship-specific short random walks in the network. As in various homogeneous representation learning approaches, random walks are used to preserve the contextual relationships between nodes [26, 28, 46]. Random walk sequences are analogous to sentences. We train a sequence classifier on 1D-Convolutional

Neural Network (CNN) model. The goal is to learn a node embeddings that enable accurate sequence classification and at the same time learn embeddings that preserve the semantic relations encoded in the multi-modal walks. Various works show that CNNs perform quite well in various NLP tasks such as sentiment classification [71] and answer selection [72]. We adopt the model from the work of Kim [71] on sentence classification due to the stark resemblance of the inputs and optimization objective. The model is a $1D$-CNN with multiple filters taking embedded vectors of random walk sequences as input.

The rest of the chapter is organized as follows: Section 7.1 briefly describes HINs, and formally defines the problem. In Section 7.2, we describe the data preparation step (7.2.1) and the design of the model (7.2.2). Experiments are described in Section 7.3.

7.1 Problem definition

Given a heterogeneous information network G_h, we aim to learn a function $\Phi : V \rightarrow \mathbb{R}^d$ which maps each node $v \in V$ to a low d-dimensional vector space $\mathbb{R}^d$, where $d << |V|$, such that the different structural relationships between nodes are preserved.

7.2 Model

In this section, we discuss the methods for generating the sequences of nodes (Section 7.2.1) followed by the description of our CNN model (Section 7.2.2).

7.2.1 Sequence Generation and Labeling

Algorithms based on language modeling require a corpus of words and sentences. Therefore, in order to use such models on networks, we need to build a corpus of sequences of vertices from the graph. Approaches based on random walks have been widely used in homogeneous networks to generate such corpus [26]. In case of heterogeneous networks, Sun et al. have shown that random walks ignoring node and edge types are highly biased towards the types associated to nodes with high degree [73]. Meta-path based random walks have been proposed [56, 58] for generating the walk sequence to capture the heterogeneous node contexts. These random walks are guided by a meta-path scheme. Meta-path based random walks require a large number of walks per node and longer walks in order to capture the contexts of nodes.

Considering these issues, we propose a scheme based on edge-constrained random walks. Given an edge type $t_i \in T_E$, a random walker traverses the paths based on the edge type t_i to generate a sequence of nodes up to a length of l. Each sequence of nodes capture the semantic and structural relationships between the nodes in HIN. For each node in an edge type, we perform multiple random walks, and we collect all random walks to form a sequence corpus S. For example, in the U.S. PATENTS heterogeneous network, we use PP - *patent-patent*, PA - *patent-assignee* , and PI - *patent-inventor* relationships to generate the sequences. The edges which connect two different types of nodes form a bipartite structure. Random walks on bipartite graphs to obtain structural contexts has already been used [46].

Labels of sequences are assigned based on labels of nodes, given that sequences form a coherent structural and semantic context. We randomly select a fraction of nodes from V, and use their labels to form a label of

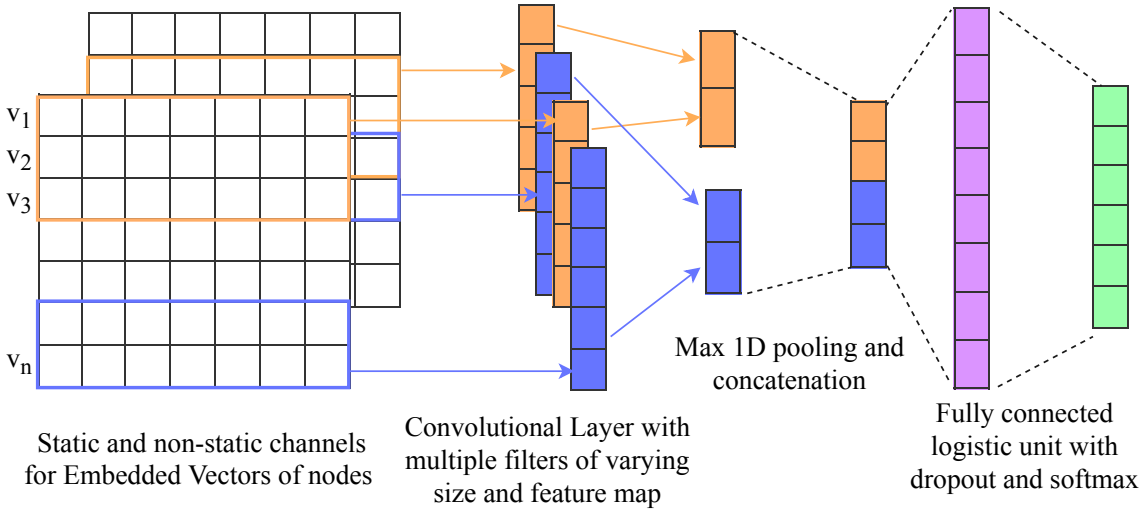

Figure 7.1: The adopted 1D-CNN Architecture for Representation Learning in HIN

a sequence. Let $U \subseteq V$ be a randomly selected fraction of nodes from V, and $\lambda : V \rightarrow \mathbb{L}$ is a label assignment function that assigns a label to a node v from a set of labels $\mathbb{L}$. The label for a sequence s is:

$$L_s = [l : \lambda(v), \forall v \in s \; and \; v \in U]$$

7.2.2 HetNet2Vec Model

We adopt the CNN model proposed by Kim [71] for representation learning of nodes. The architecture of the model is shown in Fig. 7.1. The model was proposed to solve a sentence classification problem. The selection of this model is motivated by the stark resemblance of our data input *vs* the model input and the optimization objective function. Instead of the words in sentence classification, we have nodes and the sequence of nodes represents a sentence. The sequences of nodes are obtained by edge-guided random walks as described in Section 7.2.1. The model classifies the sequences while optimizing the representation of the nodes.

The model input is a sequence S_i of vertices of length n, padded if necessary. Each vertex $v \in S_i$ is represented by an embedding vector

$\mathbf{e}^j = \Phi(v) \in \mathbb{R}^d$. A sequence is represented as a matrix $\mathbf{M}$ of embedding vectors of its vertices as $\mathbf{M} = \left[\mathbf{e}^1, \mathbf{e}^2, \cdots, \mathbf{e}^n\right]$. Let $\mathbf{M}[i:j]$ represent the concatenation of row vectors from i^{th} row to j^{th} row. In the convolutional layer, we apply multiple filters of different sizes, $\mathbf{W}^k \in \mathbb{R}^{hd}$ where h is the height of the k^{th} filter, to produce a new feature. The feature $\mathbf{c}_k[i]$ is obtained by applying the k^{th} filter on the sub-matrices of $\mathbf{M}$ as:

$$\mathbf{c}^k[i] = f(\mathbf{W}^k \cdot \mathbf{M}[i : i + h - 1] + b) \tag{7.1}$$

where $i \in [1, \ldots, n - h + 1]$, f is non-linear activation function, such as *relu*, $\cdot$ is the dot product between the filter and the concatenated vectors of matrix $\mathbf{M}[i : i + h - 1]$, and b is the bias. The convolutional operator is applied multiple times to obtain a sequence of feature maps $\mathbf{c}_k \in \mathbb{R}^{n-h+1}$. In case of a multi-channel architecture as shown in Fig. 7.1, each filter is applied to both channels and the respective feature maps are added.

Multiple feature maps are produced from varying filter sizes. The dimensionality of each feature map varies according to the filter size. We apply *1-max-pooling* [74] to obtain fixed length vectors from each feature maps. These vectors are concatenated to form a top-level feature vector which is followed by a fully connected softmax layer which produces a probability distribution over labels. At the dense layer, we apply dropout regularization as proposed by Hinton et al. [75]. The training objective of the model is to minimize the binary cross-entropy while optimizing the weight vectors. In case of the two-channel architecture, both channels have node vectors but one is kept static throughout the training and other is tuned via back-propagation. The parameters include embedding weights, layer weight, and b bias term are trained using the back-propagation method.

Dataset	Nodes			Edges	Labels
USPATENTS	Patent	Inventor	Assignee	253,537	14
	63,486	83,893	10,245		
Yelp	User	Business	City	168,701	10
	22,073	5914	10		

Table 7.1: Dataset Statistics

7.3 Experiments

In this section, we present an evaluation of our proposed approach. We first provide the description of two real-world datasets and the state-of-the-art baseline methods. Then, we provide an experimental setup for evaluation through multi-class classification of nodes.

7.3.1 Datasets

We use two heterogeneous datasets for evaluation: U.S. Patents[1] and Yelp[2]. Table 7.3.1 summarizes some statistics of these datasets. Edges between nodes can be either undirected ($-$) or directed ($\rightarrow$).

U.S. Patents is a 3-year (1998-2000) drug[3] related patent dataset obtained from United States Patent and Trademark Office. The network contains three different types of nodes, patent (P), Inventor (I) and Assignee (A). The network has patent citations $P \rightarrow P$, patent-inventor $P - I$, and patent-assignee $P - A$ as edges.

Yelp is a dataset on reviews of restaurants by customers. We extracted data of one year (2010) of restaurants serving at least one of the 10 cuisines[4].

[1] `http://www.patentsview.org/download/`

[2] `https://www.yelp.com/dataset/challenge`

[3] Drug patent classes: 128, 351, 424, 433, 435, 514, 600, 601, 602, 604, 606, 607, 800

[4] American, Mexican, Italian, Chinese, Japanese, Thai, Indian, Canadian, Spanish, Greek

The dataset has three types of nodes: restaurants (R), users (U), and city (C). There are three types of relationships between nodes: user friendships, $U - U$, user's reviews, $R - U$, and restaurant's city $R - C$.

7.3.2 Baselines

We evaluate our approach against state-of-the-art NRL methods. Since DeepWalk and Node2Vec methods are designed for a homogeneous network, we will treat all nodes and edges in heterogeneous network as homogeneous ones to learn an embedding in both these approaches. The results from the baselines are obtained using the code released by the authors.

- DeepWalk [26] is a random-walk based method to learns a d-dimensional node vectors.
- Node2Vec [28] is a parameterized random walk based approach to generate the vertex sequences. The node embeddings are learned by using negative sampling in SkipGram.
- Hin2Vec [58] is a single-hidden-layer neural network model to learn an embedding of nodes using meta-path based relationships between different nodes, and also learns an embedding for meta-paths in the heterogeneous networks.

7.3.3 Experimental Setup

For the corpus generation we performed 10 random walks on each node in each relation, with a length of 80. For labeling the sequences, we used the labels of 50% of the nodes. Then we randomly selected a portion of these sequences for training. In case of a two-channel architecture, we initialized the non-static layer node vectors with the pre-trained word2vec vectors,

Metric	Method	Micro-F1	Macro-F1
U.S. Patents	DeepWalk	0.47	0.39
	Node2Vec	0.47	0.38
	Hin2Vec	0.49	0.41
	HetNet2Vec	**0.53**	**0.43**
	HetNet2Vec$_{rand}$	0.51	0.42
Yelp	DeepWalk	0.18	0.35
	Node2Vec	0.17	**0.36**
	Hin2Vec	**0.19**	**0.36**
	HetNet2Vec	0.18	0.34

Table 7.2: Multi-class Classification on Patents and Restaurants nodes

and the static layer is initialized randomly. In a single-channel architecture, the node vectors are randomly initialized. The hyper-parameters for the model such as number and size of filters, number of epochs were selected by grid search.

The dimensionality of node vectors is set to 128 in all methods. The hyper-parameters for the baselines are selected as reported in the respective papers to get better performance.

7.3.4 Classification

After learning the node representation on the entire dataset, we perform patent and restaurant node classification in U.S. Patents and Yelp datasets, respectively. In each dataset, we selected the nodes which were used in training the CNN model and use their representations as feature vectors for classification. These nodes comprise the training data and the remaining nodes are the test set for classifier. We trained a one-vs-rest logistic regression classifier, Liblinear [68]. We report the average Micro-F1 and Macro-F1 scores as metrics for evaluation.

The results of node classification are shown in Table 7.2. In U.S.

PATENTS dataset, HETNET2VEC shows the improvement over the state-of-the-art models relatively by $8\% - 12\%$ and $4\% - 10\%$ in terms of MICRO-F1 and MACRO-F1, respectively. This indicates that the proposed model is able to preserve the different relationships in the HIN. Moreover, HETNET2VEC$_{ra}$ which is a single channel variant and the node vectors are initialized randomly, also performs better. In case of YELP dataset, the results are not too far from the baselines. This warrants further investigation in the model architecture and hyper-parameters.

Chapter 8

Predicting Virality of Cascades

On social networks platforms, content is usually diffused over the underlying social graph that represents the connections among the users in these frameworks. Early research on predicting cascade virality assumed strong correlations between the spread of content and structural properties of users who started these 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 [7, 76–80]. 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 world cup period), 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 differ-

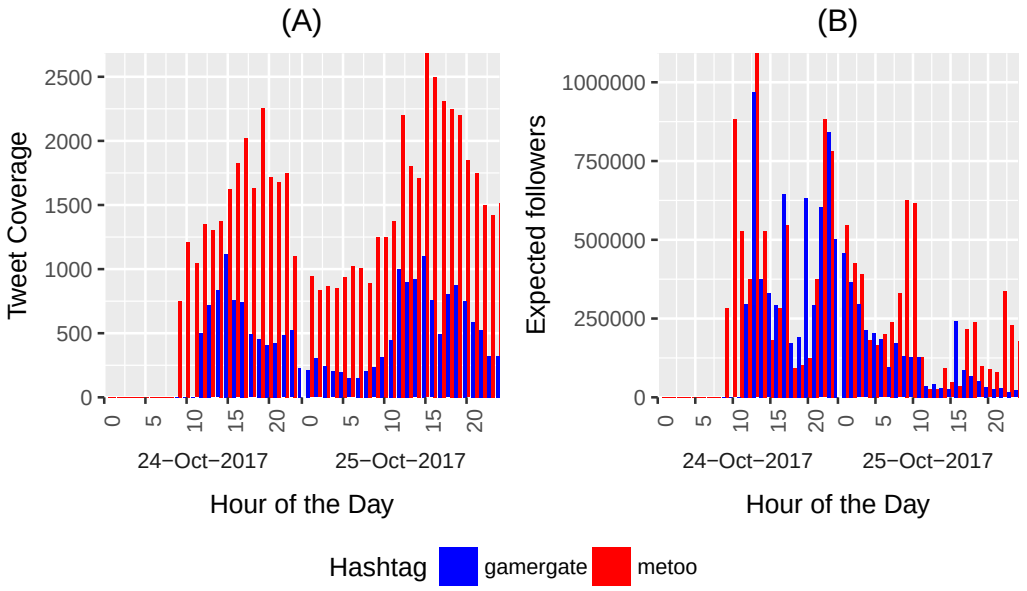

Figure 8.1: Examples of two recent hashtag campaigns. (A) The tweeting frequency of each hashtag; `#metoo` achieved more spread compared to `#gamergate`. (B) The network properties of the participating nodes in each hashtag in terms of average number of followers; the nodes engaged in the first 12 hours almost achieve similar reachability in both hashtags.

ent spread patterns even when considering similar network properties of the engaged nodes in the underlying social graph; thus, network properties may not be the best or the only indicator for virality. For example, Fig. 8.1(A) shows the spread patterns of two hashtag campaigns `#metoo` and `#gamergate` that happened almost at the same time[1]. 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[2]. 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), as shown in Fig. 8.1(B).

[1]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

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

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 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 [81].

For the reasons above, it becomes important to design algorithms that do not require any type of features or information about the underlying network, but are still able to effectively predict cascade virality in the very early stages of the spreading. Some initial but also effective attempts towards exploring this *network-agnostic* approach have already demonstrated the potential for effective and timely prediction based only on information that could be learned from the cascades themselves without requiring any other additional information [82]. However, most of the works available in the literature are mainly adopting either *"network-aware"* or at best *"quasi-network-agnostic"* approaches [8], relying on "less expensive" structural information, such as node degrees.

In this chapter we propose a novel *network-agnostic* algorithm that predicts cascade virality based only 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 should be 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. 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 time series, where each element of the series is the reaction time measured from the source signal. Furthermore, our work is partly inspired by iSAX [83], that is used for indexing time series data. In particular, 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 time series of a cascade is transformed into discrete values by using equally-sized periods of times.

For the actual prediction task, we adopt a technique from the Natural Language Processing (NLP) community that has been used for sentence classification [84]. The algorithm exploits a deep convolutional neural network (CNN) model to effectively predict sentences; instead of sentences, we feed the neural network with the transformed time series. The choice of this model is inspired by an empirical observation that shows a similarity between the distribution of words appearance in sentences and the distribution of discretized values appearance in cascades (time series). Furthermore, CNNs achieve performances as good as RNNs, which are a natural fit in such settings, but they can be trained more easily.

Summary of contributions

CAS2VEC provides a novel network-agnostic approach that models information cascades as time series and discretizes them using time slices. Furthermore, CAS2VEC can predict whether a cascade is going to become viral or not based only on the timestamps of the events encoded in the cascade itself. To show the effectiveness of our algorithm in cascade prediction, we have performed extensive experiments and compared it against strong baselines. Our results show that CAS2VEC outperforms them by an increase between 10% and 20% in all the tasks.

The rest of the chapter is organized as the following. Section 8.2 briefly describes some basic cascade definitions and assumptions that are related

to our model. Section 8.3 illustrates the design of our algorithm. We discuss the experimental evaluation of CAS2VEC against strong baselines in Section 8.4. Then, Section 8.1 briefly describes the state-of-the-art methods used for cascade virality prediction.

8.1 Related work

Many research works have dedicated effort to the prediction of web content popularity with the focus on achieving (i) good predictions (ii) in the shortest possible time windows and (iii) using the least possible information. Related research predicts cascades development either in terms of the potential size they can grow to (i.e., regression approach [8, 85–87]), or in terms of classifying them as viral or not-viral (i.e., classification approach [76, 86, 88, 89]). In both approaches, most works are based on either topological information or on features such as temporal properties, structure of the cascade at its first stage, the content in question, the early adopters, etc.

Other studies, however, have utilized little or no network information [8, 82, 90–92]. Recent studies predict content popularity based on *point process models* and node degree [8, 92], whereas another study uses survival analysis technique and follows a network-agnostic approach [82]. Under the regression approach, some works have taken the direction of predicting the optimum future size of a cascade [8], whereas others have provided time-based predictions of the cascade growth function [87]. Regardless of the approach taken, most works have been based on either topological information or on features such as temporal properties, structure of the cascade at its first stage, the content in question, the source or key early adopters, etc.

On one hand, some works have based on generative models of these fac-

tors as distributions or stochastic processes that interpret the event series in the cascade [87, 93]. On the other hand, other works based on representative models through handcrafted and heuristic-based features that are mainly extracted from knowledge about the domain and the content in question. These features are integrated using discriminative machine learning algorithms that can be used to achieve either the classification or the regression tasks [76, 86, 88].

In our study, we adopt a fully network-agnostic and domain insensitive approach, where only information available in the cascade is being deployed. We investigate a complementary approach using deep CNNs. A related approach has been taken by a very recent work [94], however with different cascade modeling schemes and classification algorithms.

8.2 Model and definitions

Cascades naturally capture 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 underlying connectivity of users, and we will simply consider a cascade as a sequence of events.

A *sequence of events* S is represented by the ordered list of *timestamps* at which the events occur:

$$S = [t_1, ..., t_s]$$

Timestamps are measured relatively to the first event of the sequence, so $t_1 = 0$. Since events are ordered, we have that $i < j \Rightarrow t_i < t_j$.

We use $S[i] = t_i$ to denote the timestamp of the i^{th} event. We write $t \in S$ to denote the fact that t corresponds to an event that has been included in S, i.e. $S[i] = t$ for some i between 1 and $|S|$, the length of the sequence.

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

$$S(t_b, t_e) = [t : t \in S \wedge t_b \leq t < t_e]$$

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

$$S(t_e) = S(t_1, t_e)$$

A *cascade* is an actual sequence of share events recorded from an online social network; the set of cascades $\mathcal{C} = \{S_1, S_2, S_3, \cdots, S_m\}$ are available as input of our problem.

There are several prediction tasks that can be computed over cascades; in this work, we will focus on *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 one of the most important issues [82].

To formally define the virality prediction problem, we consider the subsequence $O = C(t_o)$ of events occurring from the beginning of the cascade C up to an *observation time* t_o. We call such subsequence an *observation* of C; the period of time between 0 and t_o is called the *observation window*.

Given an observation $O = C(t_o)$, a *prediction window* is period starting from time t_o and lasting Δ time units, after which we want to establish whether a specific cascade is going viral or not. For this purpose, we consider the number of events $|C(t_o + \Delta)|$ that have occurred in C by the *prediction time* $t_p = t_o + \Delta$.

Similar to existing studies [7, 82], we consider two ways of determining whether a cascade is viral or not:

- through an absolute threshold $\theta_a \in \mathbb{R}^+$, the cascade C is viral if $|C(t_o + \Delta)| \geq \theta_a$;

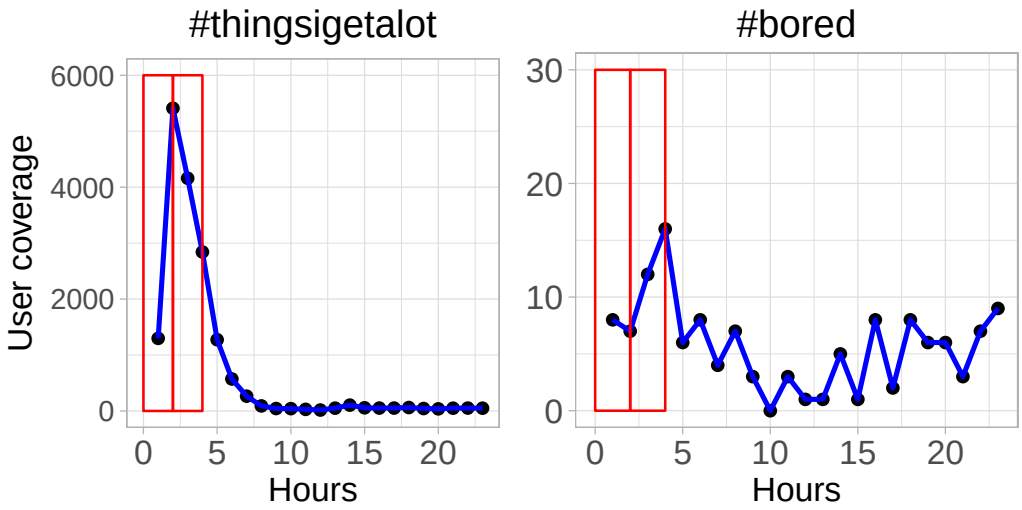

Figure 8.2: Two slices of size 2 hours, applied to the user coverage distribution of a viral hashtag (`#thingsigetalot`) and non-viral hashtag (`#bored`), which have reached 13711 and 43 users in an observation window size of 4 hours.

- through a relative threshold $\theta_r \in (0, 1)$, the cascade C is viral if $|C(t_o + \Delta)| \geq |perc(C, \theta_r)|$, where $perc(C, \theta_r)$ is the θ_r-percentile among the cascades in $\mathcal{C}$.

Our problem is thus the following: we seek to predict whether a cascade C is going to be viral by the prediction time $t_p = t_o + \Delta$, by inspecting its observation $C(t_o)$.

8.3 cas2vec

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. 8.2 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 [8, 92, 95] start from the above assumption 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. 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 8.3.1, this is a strong signal for potential virality.

From a high-level point of view, our solution is organized as follows. For each cascade C in our training data set $\mathcal{C}$, we perform three operations:

- we extract the observation $C(t_o)$, where t_o is the observation time at which the observation period ends and the prediction starts;
- we pre-process the observation $C(t_o)$ by transforming it into a format that can be fed to our classification task;
- we label the cascade C as viral or not viral, based on the threshold θ according to the number of events observed at time $t_o+\Delta$, as discussed in the previous section.

Using the transformed sequences and their associated labels, we train our classifier based on an 1D convolutional neural network.

8.3.1 Preprocessing Cascades

The observation period is divided into a collection of *slices*, i.e. equally-sized time windows. For example, Fig. 8.2 illustrates an observation window of 4 hours, divided in two slices of 2 hours each, visualized through red

boxes. Slices are identified by the *slice size* t_s; 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:

$$C^c = [\ |C(i \cdot t_s, (i+1) \cdot t_s)| : 0 \leq i < N_s\ ]$$

Discrete sequence the sequence generated by discretizing all the events within each slices, *i.e.* by assigning each event within a slice the index of the slice itself.

$$C^d = [\ \lceil C[i]/t_s \rceil : 1 \leq i \leq |C|\ ]$$

For example, look again at Fig. 8.2 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 $C_1^c = [6\,709, 7\,002]$ and $C_2^c = [15, 28]$; 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 are just 15 and 28. The discrete sequences are equal to:

$$C_1^d = [\overbrace{1, \ldots, 1}^{6\,709}, \overbrace{2, \ldots, 2}^{7\,002}]$$

$$C_2^d = [\overbrace{1, \ldots, 1}^{15}, \overbrace{2, \ldots, 2}^{28}]$$

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

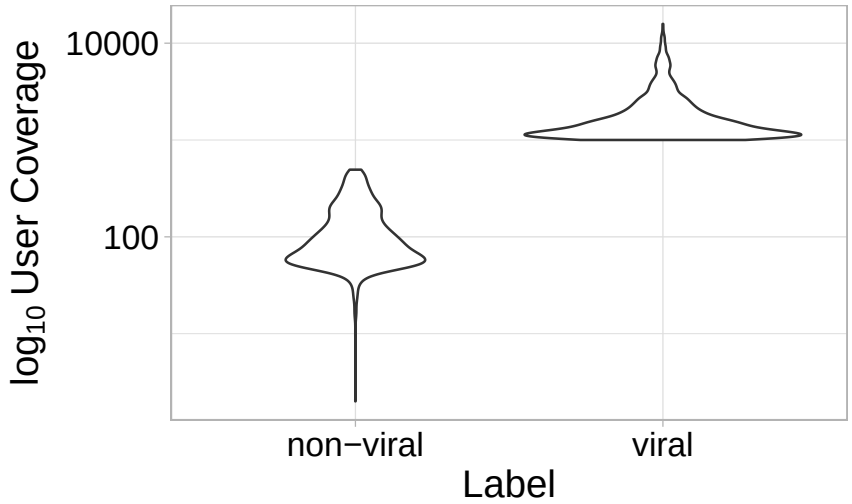

Figure 8.3: The distribution of the user coverages for the viral and non-viral classes. The user coverage distribution is computed at observation time t_o as $|C(t_o)|$ and virality is computed at prediction time $t_o + \Delta$. A cascade is viral if $|C(t_o + \Delta)| \geq 1,000$ and not-viral if $|C(t_o + \Delta)| \leq 500$

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

Conjecture 1. *Consider two cascades C_1 and C_2 and an absolute threshold θ. Given an observation t_o and a prediction windows 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 $|C_1(t_o)| \gg |C_2(t_o)|$.*

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. 8.2, we have 13 711 events for the viral hashtag `#thingsigetalot` and just 43 events for the non-viral hashtag `#bored`. More generally, the user coverage distribution for the two classes, shown in Fig. 8.3, further establishes an empirical case for the conjecture.

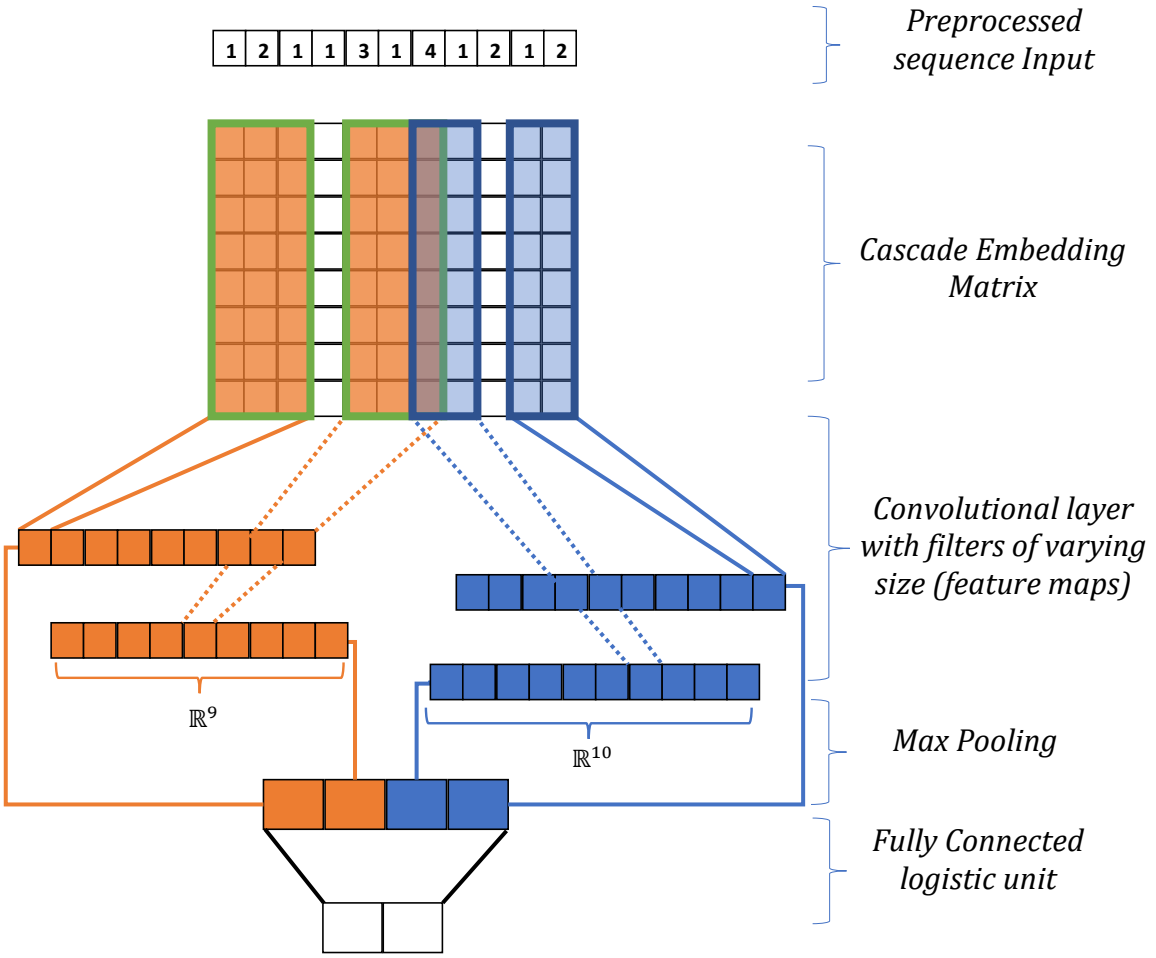

Figure 8.4: The CNN model adopted for cascade prediction

8.3.2 CNN model for cascade prediction

Once cascades are pre-processed using slices, we adopt the CNN model [84] to predict whether they will go viral or not.

The architecture of the model [84] adopted for cascade prediction is shown in Fig. 8.4. Originally this model was proposed for sentence classification in natural language documents, and it has been shown to be effective for this classification task. In addition, our choice is motivated by recent studies that have shown the CNN-based models outperform existing state-of-the-art techniques in time-series classification tasks [96, 97].

For the sake of being self-contained, we give an overview of the model;

however, because of space limitations, we will be restricted to a brief description sufficient enough to replicate our results. Interested users are referred to the original paper [84]. Instead of words in sentence classification, we have the discretized values (numbers) obtained by transforming the sequences as shown in Section 8.3.1. The input of the model is a pre-processed sequence C_i (e.g., a sequence of counter, labeled as *pre-processed sequence input* in Fig. 8.4). Each input c_i in the sequence is represented by an embedding vector $\mathbf{c_i} \in \mathbf{R}^d$. The entire sequence is denoted by a matrix, referred to as *cascade embedding matrix* in Fig. 8.4, and it is denoted by $\mathbf{M} = [\mathbf{c_1}, \ldots, \mathbf{c_s}]$ where s is the length of the sequence.

Assume the model that we are going to describe is trained and all its parameters are tuned to their optimal values. Then, the prediction task starts by applying a set of p filters on the cascade embedding matrix in the convolutional layer. That is, we apply p filters (denoted by different colors) of different sizes on every possible slice of the input (the cascade embedding matrix). More formally,

$$\phi_l = f(\mathbf{w_i} \cdot \mathbf{m_k} + b)$$

where the vector $\mathbf{w_i} \in \mathbb{R}^{kd}$ is the i^{th} filter, $b \in \mathbb{R}$ is the bias, f is an activation function, such as *relu*, k is the size of the i^{th} filter, and ϕ_l is the l^{th} feature value. $\mathbf{m_k} = \mathbf{M}[j] \oplus \ldots \oplus \mathbf{M}[j+k] \in \mathbb{R}^{kd}$ is a concatenation of the k–columns of the matrix $\mathbf{M}$. Generally, the i^{th} filter of size k is applied $s - k + 1$ times, to give a feature map $\boldsymbol{\phi}_i = [\phi_{i,1}, \ldots, \phi_{i,s-k+1}]$. $\boldsymbol{\phi}_i$ captures patterns in high-level features, such as *n–grams* in language documents. In our setting this corresponds to patterns within small subsequences depending on the filter size.

Next, a max-pooling (or a max-overtime-pooling) operation is applied over each feature map, which is simply a $\max(\boldsymbol{\phi}_i) = \widehat{\phi}_i \in \mathbb{R}$ of each feature map $\boldsymbol{\phi}_i$. 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 logistic classification layer. The vector $\mathbf{z}$ can be viewed as the final set of features extracted for the current cascade, and it will be used to predict the cascade into one of the two classes $y = \{1 = \text{viral}, 0 = \text{non–viral}\}$.

Training the Model 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 pre-processed sequences. More formally, we adopt the standard binary cross-entropy objective function:

$$\min \sum_i y_i \log(h(S_i)) + (1 - y_i) \log(1 - h(S_i))$$

Here, S_i and $y_i \in \{0, 1\}$ are the i^{th} pre-processed sequence and class label, respectively. h is the proposed model that produces a probability distribution (prediction) $\mathbf{y}$ for the given input sequences S over the classes (viral and non-viral):

$$\mathbf{y} = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{v}) + b$$

where $\mathbf{v}$ is a Bernoulli distribution used for dropout regularization as proposed in [84].

Ultimately, the model parameters $[\mathbf{M}, b, \mathbf{w_i}, \mathbf{w}]$ are trained using the back-propagation algorithm.

8.4 Experiments and Results

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.

8.4.1 Datasets

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

- *Twitter*: This dataset has been widely used for cascade prediction [8, 82]. 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 [79, 80].

8.4.2 Baselines

We have compared our algorithm against three competing approaches; nevertheless, some well-known baselines have not been included, because their source code is not available [82].

- SEISMIC: This is a recent, state-of-the-art study that predicts the popularity of tweets using a *self-exciting point process* model [8]. It estimates the infectiousness of a tweet at time t, based on the number of reshares 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 [82] 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 [3].

- Logistic Regression (LOR): This baseline has been used in previous studies [76, 82]. We use a set of features $X = [x(1), \ldots, x(N_s)]$ com-

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

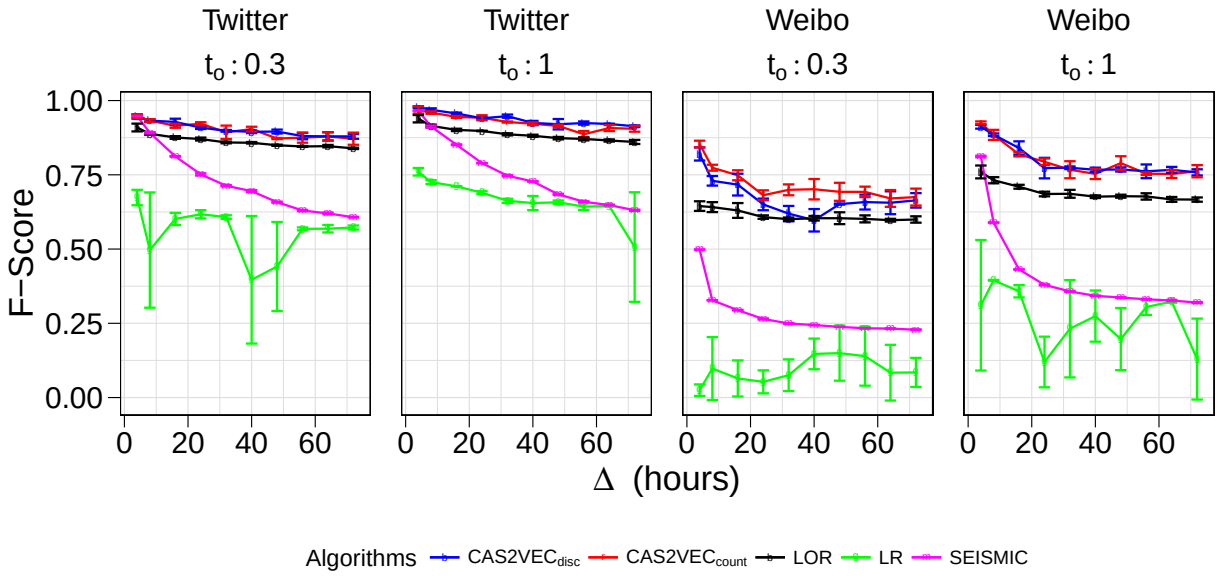

Figure 8.5: Virality prediction results for both of our datasets. For Twitter, *filter sizes* = 3, 5, 7 and for each filter we have 16 of them. For Weibo, *filter sizes* = 2, 4, 5, 7 and for each filter we have 64 of them. For both datasets, the embedding size d is 128, the number of units in the fully connected layer is 32, and the *number of slices* is 40.

puted based on the notion of slices in Section 8.3.1, 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 [8, 82]. It is specified as:

$$\log R_\infty = \log(\alpha \cdot R_t) + b + \epsilon,$$

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.

8.4.3 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.

We tune the hyper-parameters, e.g. the number and size of filters, using a *development set* (*dev-set*) during the training process. Once the hyper-parameters are tuned, then for all the experiments we fix the parameters at these values and evaluate the performance of algorithms. Towards this end, we have used a 3-fold cross validation that does not include the dev-set and reported the average result along with the error margins.

Similar to previous studies [82], we have used are the F-score with $\beta = 3$ (since it is a rare class prediction) and recall as the evaluation metric . In all the experiments, the threshold for labeling cascades is $\theta_a = 700$ that is equivalent to $\theta_r \approx 98\%$.

8.4.4 Results

Predicting Virality 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. Fig. 8.5 reports the evaluation results. All the variants of our algorithm (CAS2VEC$_{\text{count}}$, CAS2VEC$_{\text{disc}}$, CAS2VEC$_{\text{fusion}}$) outperform the baselines, and provide very similar results. The strongest 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 very 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 $t_o = 1$ hour 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 $t_o = 1$ hour respectively. CAS2VEC on the other hand achieves a significantly high 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. This, however, does not tell us how early can an algorithm predicts a virality.

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 [82]. 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 $t_p = t_o + \Delta$ for each of the dataset, i.e. 16 hours for Twitter and 34 hours for Weibo.

We then vary the size of the prediction window size Δ, from 1 hour to $t_p - 1$ hours and evaluate how early the algorithms can predict virality. Parameter Δ is similar to the time-to-virality parameter defined in [82]. Note that having fixed the prediction time, these means that the observa-

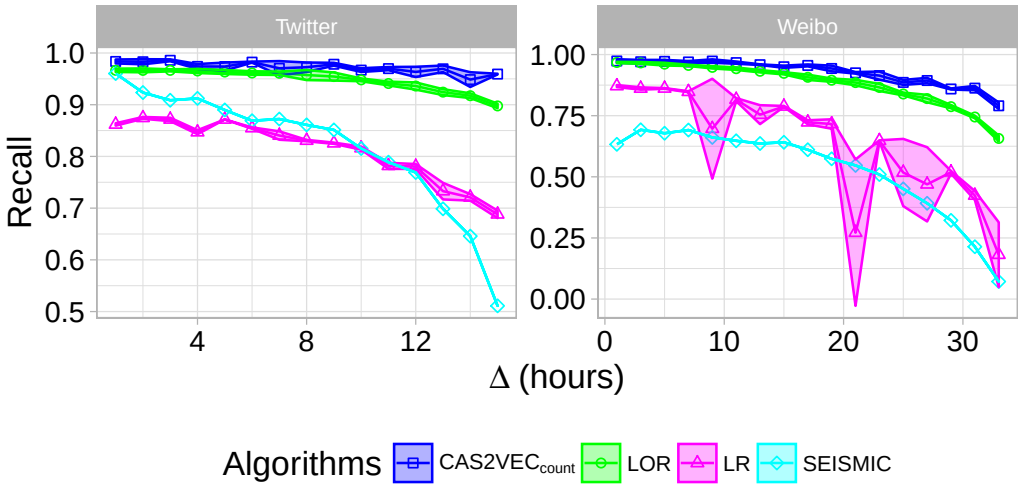

Figure 8.6: Evaluation results of early prediction experiments for the Twitter and Weibo datasets. The same hyper-parameter values as Fig. 8.5 is used

tion time t_o varies inversely w.r.t. the prediction windows size, from $t_p - 1$ hours to 1 hour. In both cases, the variation step is 1 hour.

In the following experiment (Fig. 8.6), we evaluate the recall score only for the viral calls, that is measured by the fraction of viral cascades detected by an algorithm out of all the viral cascades.

CAS2VEC obtains the best result in all of them. As one might expect, the algorithms achieve good results for small values 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.

As we get close to $\Delta = 15$ hours (Twitter) and $\Delta = 33$ hours (Weibo), which is equivalent to observing the cascade growth just for 1 hour, the performance of the baselines drop faster than CAS2VEC, which achieves the best recall. SEISMIC achieves the best results up to $\Delta = 8$, which is after observing for more than 7 and 25 hours for Twitter and Weibo respectively. However, as we go beyond, SEISMIC starts to decrease quickly; when $\Delta = 15$ (Twitter) and $\Delta = 34$ (Weibo), it recalls only 86% and 42% of the viral

cascades, respectively. The other strong baseline, LOR, at the end points drops to 89% and 56% of recall, while CAS2VEC outperforms the baselines and gets to 95% and 62% for Twitter and Weibo respectively.

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.

Break-out Coverage One of the important tasks in cascade prediction is detecting break-out events. Towards this end, similar to [8, 82], 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. 8.7-8.8. Yet again, CAS2VEC consistently achieves a significant performance gain, specially as Δ increases. Note 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. Similar to the previous experiment, it is important to achieve a high coverage as we increase Δ. In particular, 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%, where as CAS2VEC achieves a more than 90% for $k = 10$ and 20, respectively.

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. 8.9, the performance increases as we increase the number

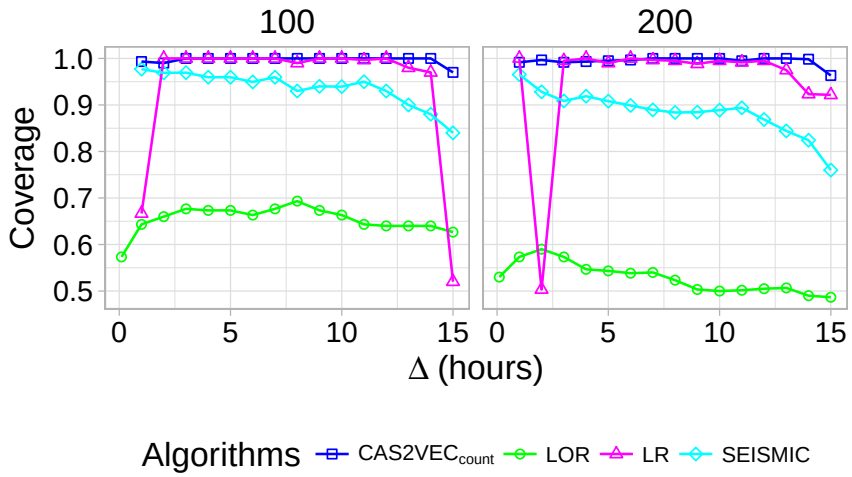

Figure 8.7: Break-out coverage for $k = 100$ and $k = 200$ for the Twitter dataset.

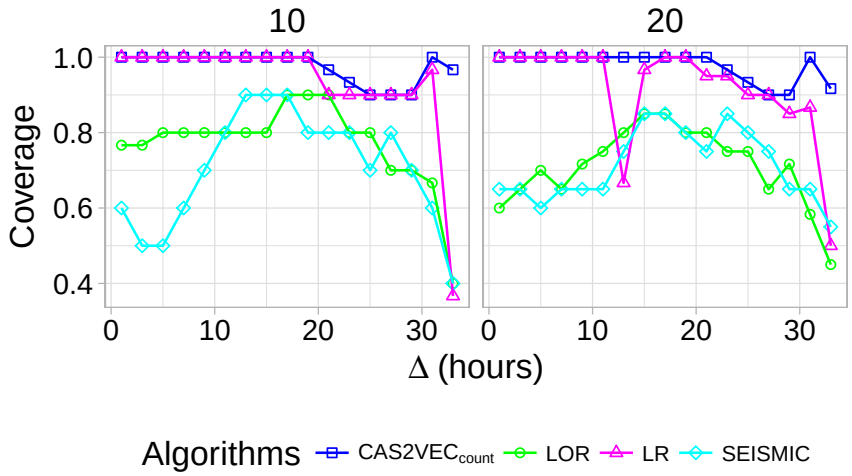

Figure 8.8: Break-out coverage for $k = 10$ and $k = 20$ for the Weibo dataset.

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. Whereas in 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. 8.10 show the effect of sequence length (determined by t_o) for the two variants of our algorithms. Particularly CAS2VEC$_{\text{disc}}$ requires more time to finish an

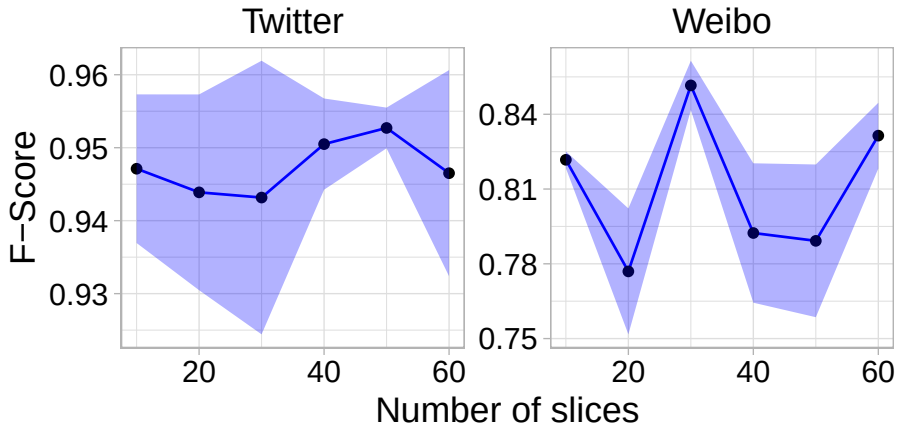

Figure 8.9: Effect of the number of slices on virality prediction at $t_o = 1$ hour and $\Delta = 12$ hours.

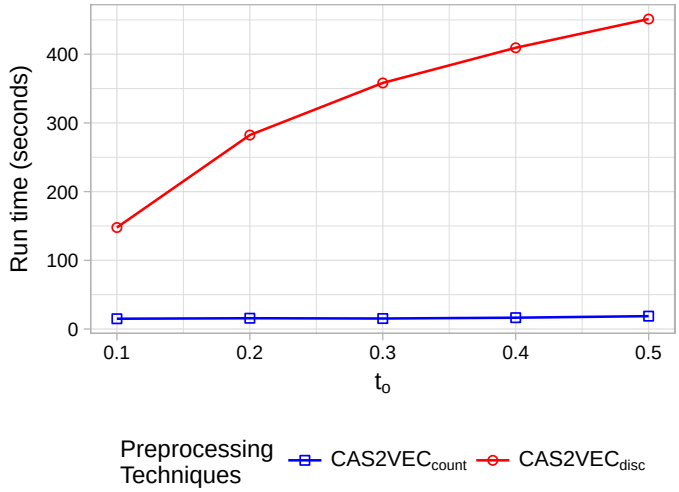

Figure 8.10: Effect of sequence length on running time.

epoch as we increase the sequence length. However, Fig. 8.11 shows that increasing the sequence length beyond a certain value (150 in the figure) does not give significant performance gain.

This chapter presents CAS2VEC, a novel algorithm for the prediction of the virality of social network cascades. Traditionally, network structure parameters and features extracted from the underlying domain have been used to perform the prediction either as a regression or a classification tasks. However, network information and predictive features are expensive

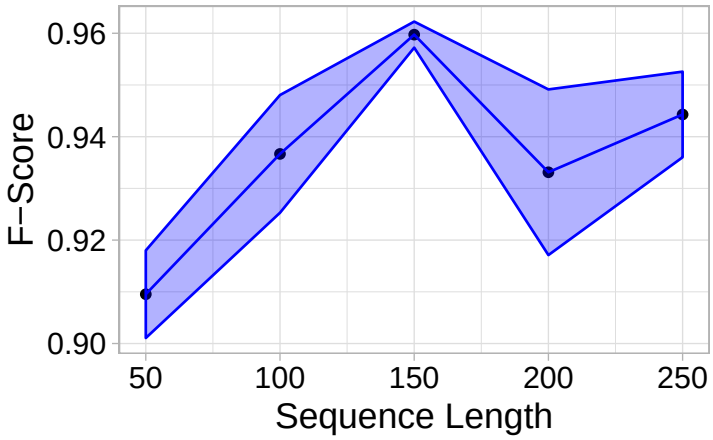

Figure 8.11: Effect of seq. length on virality prediction.

to get either because of privacy constraints or need for domain knowledge and awareness on external affecting factors.

Unlike previous work, our approach is fully network-agnostic: it is solely based on timing information explicitly encoded in the cascade. We make use of state-of-the-art techniques for sentence classification using CNNs for the actual prediction over time series. Our experiments show that time sequences in cascades are sufficient to make timely and accurate virality predictions.

CAS2VEC achieves an increase in prediction accuracy between 10% and 20% in all the tasks w.r.t. F-score and recall compared to strong baselines in the field, while being fully network-agnostic. As future work, we plan to extend the model by incorporating extra features like content, early adopters, and analyze the interpretability of the learned features.

Chapter 9

Conclusion

The ubiquitous nature of graphs in vast and diverse domains such as biology, economy and technology provides an opportunity to get greater insights into these areas and use modern machine learning approaches to build useful applications. Therefore, it is essential to obtain relevant and useful features from them. This thesis is a step towards this direction; we have built models which are good at exploiting the diverse kinds of information present in the graphs for learning representation, which are then further used in downstream machine learning applications. We focused on node classification, link prediction, and nearest-neighbor tasks.

Most real-world graphs are associated with rich content information such as attributes. It is challenging to incorporate attribute information in learning. We proposed two different models to exploit the attributed graphs to learn a better representation. In our first contribution, we proposed GAT2VEC that uses structural information and attribute information to generate structural contexts and attribute contexts respectively, through random walks. Then using these two different contextual information jointly helps in learning a better representation by preserving the structural and attribute contexts of a node which is empirically shown through machine learning tasks.

SAGE2VEC is our second contribution towards representation learning on attributed graphs. As observed, the structure of the graph is highly non-linear and sparse, and this problem is aggravated in the case of attributed graphs. The contribution of this work is to provide a simple model which handles non-linearity and sparsity of both network structure and attributes, in addition to avoiding complex optimizations for modeling complex relationships proposed by state-of-the-art methods. We observed that a simple, well-designed model having a lower algorithmic and complexity can learn a better representation than complex baselines.

HETNET2VEC is a contribution towards the goal of learning representations of heterogeneous information networks. The work aims at obtaining node sequences such that the different semantic relationships are covered and the model aims at preserving these.

In this thesis, we also worked on prediction of virality of cascades without knowing the underlying information network. We proposed CAS2VEC, a network-agnostic approach, that models a cascade as a time-series data and train a neural network model inspired from NLP to predict virality of a cascade. Our experiments show that time sequences in cascades are sufficient to make timely and accurate virality predictions. CAS2VEC achieves an increase in prediction accuracy between 10% and 20% in all the tasks w.r.t. F-score and recall compared to strong baselines in the field, while being fully network-agnostic. As future work, we plan to extend the model by incorporating extra features like content, early adopters, and analyze the interpretability of the learned features.

The area of NRL is rapidly advancing with introduction of new models such as Graph Neural Networks (GNN), and there is lot of scope to extend it to various domains such as heterogeneous networks and dynamic networks. As in the thesis, we discussed the importance of well-designed and scalable models for large graphs. In continuation of the current focus in

NRL, the near term plan is to work on scalable GNN models in attributed and heterogeneous networks.

Bibliography

[1] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *NIPS*, 2017.

[2] W. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.

[3] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 635–644, ACM, 2011.

[4] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, pp. 355–369, Mar. 2007.

[5] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[6] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han, "Personalized entity recommendation: A heterogeneous information network approach," in *Proc. of the 7th ACM Int. Conf. on Web Search and Data Mining*, WSDM '14, pp. 283–292, ACM, 2014.

[7] L. Weng, F. Menczer, and Y.-Y. Ahn, "Virality prediction and community structure in social networks," *Sci. Rep.*, vol. 3, no. 2522, 2013.

[8] Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec, "SEISMIC: A self-exciting point process model for predicting tweet popularity," in *Proc. of KDD'15*, ACM, 2015.

[9] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc. of the 24th Int. Conf. on World Wide Web*, WWW '15, pp. 1067–1077, 2015.

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[11] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," ch. Distributed Representations, pp. 77–109, Cambridge, MA, USA: MIT Press, 1986.

[12] T. Mikolov, J. Kopecky, L. Burget, O. Glembek, and J. Cernocky, "Neural network based language models for highly inflective languages," in *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '09, (Washington, DC, USA), pp. 4725–4728, IEEE Computer Society, 2009.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[14] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proc. of the Tenth Int. Workshop on Artificial*

Intelligence and Statistics, AISTATS'05, Society for Artificial Intelligence and Statistics, 2005.

[15] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biol. Cybern.*, vol. 59, pp. 291–294, Sept. 1988.

[16] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, (San Francisco, CA, USA), pp. 3–10, Morgan Kaufmann Publishers Inc., 1993.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," pp. 318–362, Cambridge, MA, USA: MIT Press, 1986.

[18] Y. LeCun and Y. Bengio, "The handbook of brain theory and neural networks," ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258, Cambridge, MA, USA: MIT Press, 1998.

[19] M. Belkin and P. 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, (Cambridge, MA, USA), pp. 585–591, MIT Press, 2001.

[20] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *AAAI*, 2017.

[21] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proc. of the 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '09, pp. 817–826, ACM, 2009.

[22] L. Tang and H. Liu, "Leveraging social media networks for classification," *Data Mining and Knowledge Discovery*, vol. 23, pp. 447–478, Nov 2011.

[23] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, (New York, NY, USA), pp. 891–900, ACM, 2015.

[24] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 1105–1114, ACM, 2016.

[25] C. Tu, W. Zhang, Z. Liu, and M. Sun, "Max-margin deepwalk: Discriminative learning of network representation," in *IJCAI*, pp. 3889–3895, IJCAI/AAAI Press, 2016.

[26] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of KDD'14*, (New York, NY, USA), pp. 701–710, ACM, 2014.

[27] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '16, pp. 1225–1234, ACM, 2016.

[28] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proc. of the 22Nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '16, pp. 855–864, ACM, 2016.

[29] T. Lyu, Y. Zhang, and Y. Zhang, "Enhancing the network embedding quality with structural similarity," in *Proceedings of the 2017 ACM on*

Conference on Information and Knowledge Management, CIKM '17, (New York, NY, USA), pp. 147–156, ACM, 2017.

[30] J. Li, J. Zhu, and B. Zhang, "Discriminative deep random walk for network classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1004–1013, Association for Computational Linguistics, 2016.

[31] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, (New York, NY, USA), pp. 385–394, ACM, 2017.

[32] B. Shaw and T. Jebara, "Structure preserving embedding," in *Proc. of the 26th Annual Int. Conf. on Machine Learning*, ICML '09, pp. 937–944, ACM, 2009.

[33] D. Luo, C. H. Q. Ding, F. Nie, and H. Huang, "Cauchy graph embedding," in *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pp. 553–560, 2011.

[34] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 1145–1152, AAAI Press, 2016.

[35] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734 vol. 2, 2005.

[36] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *Trans. Neur. Netw.*, vol. 20, pp. 61–80, Jan. 2009.

[37] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *CoRR*, vol. abs/1312.6203, 2013.

[38] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *CoRR*, vol. abs/1506.05163, 2015.

[39] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (USA), pp. 3844–3852, Curran Associates Inc., 2016.

[40] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," *CoRR*, vol. abs/1801.03226, 2018.

[41] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *CoRR*, vol. abs/1705.07664, 2017.

[42] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proc. of the 10th ACM Int. Conf. on Web Search and Data Mining*, WSDM '17, pp. 731–739, ACM, 2017.

[43] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *SIAM International Conference on Data Mining*, pp. 633–641, 2017.

[44] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *Proc. of the 24th Int. Conf. on Artificial Intelligence*, IJCAI'15, pp. 2111–2117, AAAI Press, 2015.

[45] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Homophily, structure, and content augmented network representation learning," in *ICDM*, pp. 609–618, IEEE Computer Society, 2016.

[46] J. Tang, M. Qu, and Q. Mei, "Pte: Predictive text embedding through large-scale heterogeneous text networks," in *Proc. of the 21th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '15, pp. 1165–1174, ACM, 2015.

[47] Z. Yang, W. Cohen, and W. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *CoRR*, vol. abs/1603.08861, 2016.

[48] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence*, IJCAI'16, pp. 1895–1901, AAAI Press, 2016.

[49] H. Gao and H. Huang, "Deep attributed network embedding," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 3364–3370, 7 2018.

[50] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang, "Anrl: Attributed network representation learning via deep neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 3155–3161, 7 2018.

[51] Z. Meng, S. Liang, H. Bao, and X. Zhang, "Co-embedding attributed networks," in *Proc. of the 12th ACM Int. Conf. on Web Search and Data Mining*, WSDM '19, ACM, 2019.

[52] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[53] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[54] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pp. 2609–2615, 2018.

[55] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2013.

[56] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2Vec: Scalable representation learning for heterogeneous networks," in *Proc. of the 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '17, pp. 135–144, ACM, 2017.

[57] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, (New York, NY, USA), pp. 119–128, ACM, 2015.

[58] T.-Y. Fu, W.-C. Lee, and Z. Lei, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *Proc. of the 2017 ACM on Conf. on Information and Knowledge Management*, CIKM '17, pp. 1797–1806, ACM, 2017.

[59] Z. Huang and N. Mamoulis, "Heterogeneous information network embedding for meta path based proximity," *CoRR*, vol. abs/1701.05291, 2017.

[60] J. Shang, M. Qu, J. Liu, L. M. Kaplan, J. Han, and J. Peng, "Meta-path guided embedding for similarity search in large-scale heterogeneous information networks," *CoRR*, vol. abs/1610.09769, 2016.

[61] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, (USA), pp. 3111–3119, Curran Associates Inc., 2013.

[62] R. Hussein, D. Yang, and P. Cudré-Mauroux, "Are meta-paths necessary?: Revisiting heterogeneous graph embeddings," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, (New York, NY, USA), pp. 437–446, ACM, 2018.

[63] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, 2014.

[64] "BlogCatalog," 2017. Accessed: 2017-07-01.

[65] "dblp," 2017. Accessed: 2017-07-01.

[66] "CiteSeer," 2017. Accessed: 2017-07-01.

[67] X. Wang, L. Tang, H. Gao, and H. Liu, "Discovering overlapping groups in social media," in *Proc. of the 10th IEEE Int. Conf. on Data Mining*, ICDM 2010, pp. 569–578, IEEE, 2010.

[68] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, June 2008.

[69] J. Tang, J. Liu, M. Zhang, and Q. Mei, "Visualizing large-scale and high-dimensional data," in *Proc. of the 25th Int. Conf. on World Wide Web*, WWW '16, Int. World Wide Web Conf.s Steering Committee, 2016.

[70] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[71] Y. K., "Convolutional neural networks for sentence classification," in *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1746–1751, 2014.

[72] T. Yang, W. Yih, and C. Meek, "Wikiqa: A challenge dataset for open-domain question answering," in *Proc. of the 2015 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 2013–2018, 2015.

[73] Y. Sun, J. Han, X. Yan, P. Yu, and T. Wu, "PathSim: Meta path-based top-k similarity search in heterogeneous information networks," *PVLDB*, vol. 4, no. 11, pp. 992–1003, 2011.

[74] Y. lan Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in *IN COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2010 IEEE CONFERENCE ON*, pp. 2559–2566, IEEE, 2010.

[75] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[76] J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec, "Can cascades be predicted?," in *Proc. of WWW'14*, (New York, NY, USA), pp. 925–936, ACM, 2014.

[77] C. Li, J. Ma, X. Guo, and Q. Mei, "DeepCas: An end-to-end predictor of information cascades," in *Proc. of WWW'17*, Int. World Wide Web Conferences Steering Committee, 2017.

[78] G. Szabo and B. A. Huberman, "Predicting the popularity of online content," *Commun. ACM*, vol. 53, pp. 80–88, Aug. 2010.

[79] J. Zhang, B. Liu, J. Tang, T. Chen, and J. Li, "Social influence locality for modeling retweeting behaviors," in *Proc. of IJCAI'13*, pp. 2761–2767, AAAI Press, 2013.

[80] J. Zhang, J. Tang, J. Li, Y. Liu, and C. Xing, "Who influenced you? predicting retweet via social influence locality," *ACM Trans. Knowl. Discov. Data*, vol. 9, pp. 25:1–25:26, Apr. 2015.

[81] E. Sadikov, M. Medina, J. Leskovec, and H. Garcia-Molina, "Correcting for missing data in information cascades," in *Proc. of WSDM'11*, pp. 55–64, ACM, 2011.

[82] K. Subbian, B. A. Prakash, and L. Adamic, "Detecting large reshare cascades in social networks," in *Proc. of WWW'17*, pp. 597–605, Int. World Wide Web Conferences Steering Committee, 2017.

[83] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh, "iSAX 2.0: Indexing and mining one billion time series," in *Proc. of ICDM'10*, (Washington, DC, USA), pp. 58–67, IEEE Computer Society, 2010.

[84] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. of EMNLP'14*, pp. 1746–1751, 2014.

[85] O. Tsur and A. Rappoport, "What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities," in *Proc. of WSDM'12*, pp. 643–652, ACM.

[86] L. Weng, F. Menczer, and Y. Ahn, "Predicting successful memes using network and community structure," in *Proc. of ICWSM'14*, The AAAI Press, 2014.

[87] L. Yu, P. Cui, F. Wang, C. Song, and S. Yang, "From micro to macro: Uncovering and predicting information cascading process with behavioral dynamics," in *Proc. of ICDM'15*, pp. 559–568, 2015.

[88] P. Cui, S. Jin, L. Yu, F. Wang, W. Zhu, and S. Yang, "Cascading outbreak prediction in networks: a data-driven approach," in *Proc. of the KDD'13*, pp. 901–909, ACM, 2013.

[89] M. Jenders, G. Kasneci, and F. Naumann, "Analyzing and predicting viral tweets," in *Proc. of WWW'13*, ACM, 2013.

[90] D. Agarwal, B.-C. Chen, and P. Elango, "Spatio-temporal models for estimating click-through rate," in *Proc. of WWW'09*, (New York, NY, USA), pp. 21–30, ACM, 2009.

[91] R. Crane and D. Sornette, "Robust dynamic classes revealed by measuring the response function of a social system," *Proc. of the National Academy of Sciences*, vol. 105, no. 41, pp. 15649–15653, 2008.

[92] S. Gao, J. Ma, and Z. Chen, "Modeling and predicting retweeting dynamics on microblogging platforms," in *Proc. of WSDM'15*, (New York, NY, USA), pp. 107–116, ACM, 2015.

[93] C. Bauckhage, K. Kersting, and F. Hadiji, "Mathematical models of fads explain the temporal dynamics of internet memes.," in *Proc. of ICWSM'13*, 2013.

[94] C. Gou, H. Shen, P. Du, D. Wu, Y. Liu, and X. Cheng, "Learning sequential features for cascade outbreak prediction," *Knowledge and Information Systems*, pp. 1–19, 2018.

[95] S.-H. Yang and H. Zha, "Mixture of mutually exciting processes for viral diffusion," in *Proc. of ICML'13*, pp. II–1–II–9, JMLR.org, 2013.

[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*, vol. 28, pp. 162–169, Feb 2017.

[97] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," *CoRR*, vol. abs/1611.06455, 2016.