Understanding Graph Data Through Deep Learning Lens

Supriya Pandhre

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

भारतीय प्रौद्योगिकी संस्थान हैदराबाद

Indian Institute of Technology Hyderabad

Department of Computer Science & Engineering

June 2018

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

[illegible]

(Signature)

(Supriya Pandhre)

CS15MTECH11016

(Roll No.)

Approval Sheet

This Thesis entitled Understanding Graph Data Through Deep Learning Lens by Supriya Pandhre is approved for the degree of Master of Technology from IIT Hyderabad

[illegible]

(Manish Singh.) Examiner
Dept. of Computer Science & Eng
IITM

[illegible]

(Dr. Srijith P. K.) Examiner
Dept. Computer Science & Eng
IITH

V. B. Vineeth

(Dr. Vineeth N Balasubramanian) Adviser
Dept. of Computer Science & Eng
IITH

[illegible]

(Dr. Maunendra Sankar Desarkar) Chairman
Dept. of Computer Science & Eng
IITH

Acknowledgements

It has been a wonderful journey as a MTech student at IITH, and I was fortunate to share it with very inspiring and exceptionally intelligent people.

First, I would like to express my deepest gratitude to my adviser Dr.Vineeth N Balasubramanian, who has always been very supportive and encouraging throughout my journey. Before joining as his student, I was just a person interested in research with no clue of what exactly it means to do a research. Like a lighthouse guides a lost ship in the big ocean, he guided me. His guidance not only made me a better researcher but also a better person. Everything I could achieve in my career, its because of him.

I would like to thank Dr.Manish Gupta, with whom I was fortunate to have co-authored two papers and learned a lot from him. I would also like to thank Himangi Mittal, who has greatly helped me in getting my first research paper published at the conference. Her dedication and enthusiasm has always inspired me. A big thanksto my friends and group members: Adepu Ravi Sankar, Arghya Pal, Akilesh B, Tanya Marwah, Anirban Sarkar, Joseph K J, Arjun D'Cunha, Bhavana Jain and the list continues. Discussions with them in weekly meetings has helped me hone my research skills, making me push my boundaries and extend my thinking directions.

I am thankful to my parents who has always encouraged me to aim high and supported me in chasing my dream. Thank you for your endless love!

Abstract

Deep neural network models have established themselves as an unparalleled force in the domains of vision, speech and text processing applications in recent years. However, graphs have formed a significant component of data analytics including applications in Internet of Things, social networks, pharmaceuticals and bioinformatics. An important characteristic of these deep learning techniques is their ability to *learn* the important features which are necessary to excel at a given task, unlike traditional machine learning algorithms which are dependent on handcrafted features. However, there have been comparatively fewer efforts in deep learning to directly work on graph inputs. Various real-world problems can be easily solved by posing them as a graph analysis problem. Considering the direct impact of the success of graph analysis on business outcomes, importance of studying these complex graph data has increased exponentially over the years.

In this thesis, we address three contributions towards understanding graph data: (i) The first contribution seeks to find anomalies in graphs using graphical models; (ii) The second contribution uses deep learning with spatio-temporal random walks to learn representations of graph trajectories (paths) and shows great promise on standard graph datasets; and (iii) The third contribution seeks to propose a novel deep neural network that implicitly models attention to allow for interpretation of graph classification.

Contents

Chapter 1

Introduction

1.1 Motivation

The rising use of social networks and various other sensor networks in real-world applications ranging from politics to healthcare has made computational analysis of graphs a very important area of research today. Study of user-user interactions also plays a crucial role in applications such as user classification for targeted advertising or link prediction to suggest new connections on social networking sites. With increasing amount graph data being generated from diversity of sources, analyzing and understanding the graph data has now gained more importance than ever. In this dissertation, we consider three fundamental graph analysis tasks: anomaly detection in static graphs (arXiv 2016), learning trajectory representation on dynamic graphs (ACM CoDS-COMAD 2018) and end-to-end framework for graph classification (under review at IEEE ICDM 2018). For each of the aforementioned tasks, we developed different models and techniques using deep learning for understanding the complex graph data.

Our first contribution aims at analyzing the static graphs and finding the anomalies. Beyond graph analysis tasks like graph query processing, link analysis, influence propagation, there has recently been some work in the area of outlier detection for information network data. Although various kinds of outliers have been studied for graph data, there is not much work on anomaly detection from edge-attributed graphs. In this light, we introduce a method that detects novel outlier nodes by taking into account the node data and edge data simultaneously to detect anomalies. We model the problem as a community detection task, where outliers form a separate community. We propose a method that uses a probabilistic graphical model (Hidden Markov Random Field) for joint modeling of nodes and edges in the network to compute *Holistic Community Outliers (HCOutliers)*. Thus, our model presents a natural setting for heterogeneous graphs that have multiple edges/relationships between two nodes. EM (Expectation Maximization) is used to learn model parameters, and infer hidden community labels. Experimental results on synthetic datasets and the DBLP dataset show the effectiveness of our approach for finding novel outliers from networks.

In the quest of understanding the graph data, our second contribution addresses the analysis of temporal graphs. Analyzing the temporal behavior of nodes in time-varying graphs is useful for many applications such as targeted advertising, community evolution and outlier detection. Hence, we present a novel approach, STWalk, for learning trajectory representations of nodes in temporal graphs. The proposed framework makes use of structural properties of graphs at current and previous time-steps to learn effective node trajectory representations. STWalk performs random walks on a graph at a given time step (called *space-walk*) as well as on graphs from past time-steps (called *time-walk*) to capture the spatiotemporal behavior of nodes. We propose two variants of STWalk to learn trajectory representations. In one algorithm, we perform space-walk and time-walk as part of a single step. In the other variant, we perform space-walk and time-walk separately and combine the learned representations to get the final trajectory embedding. Extensive experiments on three real-world temporal graph datasets validate the effectiveness of the learned representations when compared to three baseline methods. We also show the goodness of the learned trajectory embeddings for change point detection, as well as demonstrate that arithmetic operations on these trajectory representations yield interesting and interpretable results.

With the unprecedented success of the deep learning techniques, particularly, Convolution Neural Networks (CNN) in computer vision, natural language processing, and speech processing domain, our third contribution seeks to extend and utilize these concepts for understanding graph-structured data. However, applying Convolution Neural Network directly on graph data presents two problems: i) variable number of nodes to process and ii) no unique way of ordering the nodes. Hence, in this work, we propose AtOrC , a novel end to end deep neural network framework that "learns to select" the important nodes as well as "learns to order" them, all in a single pass of the input graph. As the name "Attend, Order and Classify" suggests, the framework contains 3 modules: the SoftAttention module that learns which nodes to attend to, the SoftOrdering module that learns how to order them and the Convolutional Neural Network that learns local features useful for graph classification. Experiments on several standard benchmark datasets demonstrate that the proposed framework, AtOrC , achieves significant improvement on the classification task. Particularly, the algorithm was able to outperform state of the art methods with 10.66% improvement in classification accuracy on IMDB-MULTI dataset. Moreover, the attention maps generated during the classification process by the SoftAttention module can be used for interpreting the decisions made by the framework. These maps give insights into the deep neural network, making it more reliable and usable in risk-sensitive applications such as healthcare.

1.2 Contributions & Outline

The core contribution of this dissertation is three novel frameworks, each one focused on three task. The first framework detects anomalies in static graphs and hence named "HCODA":Holistic Community Outlier Detection Algorithm, the second method aims at learning graph representation on dynamic graphs and given name "STWalk": Space-Time Walk, and the final framework is designed for graph classification that learns to attend and order the graph nodes, hence named "AtOrC":Attend, Order and Classify. The details of the contributions are given below:

- HCODA: Holistic Community Outlier Detection Algorithm
  - We look at the community outlier detection problem from a holistic perspective by incorporating node data, edge data, and the linkage structure. Based on such a view, we propose the problem of detecting novel HCOutliers.

  - We model the problem as a community analysis task over a HMRF and provide inference using an EM algorithm.

- STWalk: Space-Time Walk
  - We present an unsupervised trajectory learning algorithm that embeds the spatial and temporal characteristics of nodes over a given time window. The algorithm carries out a random walk in a current graph, resulting in a *space-walk*, as well as in graphs at different time steps, called *time-walk*. The traversed random walks are then considered as sentences, with nodes as words from a vocabulary.
  - We employ the SkipGram [1] network to learn the latent representations of node trajectories such that it maximizes the probability of co-occurrence of two nodes within the specified window size.

- AtOrC: Attend, Order and Classify
  - There are three modules in our proposed framework: the Soft-Attention module, the Soft-Permutation module and the Convolutional Neural Network. As the name Attend, Order and Classifysuggests, each one performs one task: Attend, Order and Classify respectively. Each of these modules are differentiable, hence allowing backpropagation to be used to train the network. To the best of our knowledge, our proposed framework is the first end-to-end architecture that learns which nodes to focus on and how to order those nodes, in order to help the convolutional layers learn better features and make more accurate decisions on the graph classification task.
  - **Soft-Attention Module:** Our framework uses Soft-Attention over the nodes of the graph to select few important nodes. It is achieved by applying 1D Gaussian filter, whose mean and variance are learned by backpropagation algorithm. We also show that these learned attention maps can be used to interpret the decisions made by the model(figure 5.1).
  - **Soft-Permutation Module:** We also propose the Soft-Permutation network that learns the best ordering of nodes that will help the model to perform better at graph classification task.
  - We use the Convolutional Neural Network to learn the local features of the nodes. Instead of selecting only one filter size for the convolution operation, we used Inception Module, inspired by the InceptionNet [2] architecture.

Thesis Outline:

The thesis is divided into six chapters. In Chapter 1, we introduce the graph analysis and provide overview of our contributions. In Chapter 2, we give background on graph analysis. The Chapter 3 presents our work on outlier detection in edge-attributed graphs. In Chapter 4, we propose our contribution in learning representation on temporal graph. In Chapter 5, we present our work on graph classification. Finally, we conclude in Chapter 6.

Chapter 2

Background: Graph Analysis

2.1 Graph Outlier Detection

Outlier detection has a long history and [3, 4, 5] provide extensive overviews of popular methods; [6, 7] focus on network outlier detection methods. Our work is most related to two main sub-areas of network outlier detection: community-based outlier detection and analysis of edge-attributed graphs, each of which is discussed below.

2.1.1 Community-based Outlier Detection

Community-based outlier detection methods perform community analysis on graphs. Nodes that do not belong to any community are labeled as outliers. Community-based outlier detection has been studied both for static networks and dynamic networks. The notion of community outliers has been studied for static bipartite graphs using random walks in [8], for static homogeneous networks using probabilistic models in [9], and for static heterogeneous graphs using non-negative matrix factorization in [10]. [11] summarizes different anomaly detection methods in dynamic graphs such as decomposition based methods [12], [13], distance based methods [14]. Community based outlier detection method, for dynamic graphs, is proposed for a two-snapshots setting in [15] and for a series of snapshots in [16].

2.1.2 Analysis of Edge-attributed Graphs

Edge content indicates the type of relationship between the two nodes. However very little work exists on analysis of edge-attributed graphs. Qi et al. [17] proposed a edge-induced matrix factorization based method for finding communities in social graph using edge content. [18] proposed a method that considers edge data of User-Likes-Pages Facebook graph for temporal analysis to find the page-like pattern. [19] proposed a method for mining coherent sub-graphs in multi-layer edges by exploiting edge content. [20] proposed a method for finding top-K interesting subgraphs matching a query template where interestingness is defined using edge weights. We propose a community-based outlier detection method for edge-attributed graphs which jointly exploits edge content along with node data and the linkage structure.

2.2 Temporal Graph Analysis

Related to the other focal subarea for this work, time-varying graphs have been studied for applications such as node classification [21, 22, 23], link prediction [24], community evolution [25] and outlier detection [7] in recent years. In one of the earlier works in this area, Tang et al. [25] proposed a spectral clustering framework for studying the evolution of communities in time varying multi-mode networks. Aggarwal et al. [21] proposed a node classification model that used network structure and node attributes from time-varying graphs. The model was specifically designed for textual node attributes. In [24], a non-parametric model was proposed for link prediction in a series of network snapshots. More recently, Li et al. [23] presented a framework that used node attributes for learning node embeddings. The model first learns separate embeddings from network structure and node attributes respectively. The node embeddings are then combined using matrix perturbation theory. Another recent work, GraphSAGE [22], proposed an inductive algorithm that generates node embeddings using sampling and aggregating local neighborhood information and learning a function to generate node embeddings of unseen data. For more such methods, Aggarwal et al. [26] provide a comprehensive survey on methods for temporal graph analysis.

While there have been a few efforts on analysis of time-varying graphs, all the aforementioned algorithms enforce the data to have node attributes. Many real-world graph data, however, may not provide node features either because they are not available publicly for use or they are missing (for instance, it is optional to fill age or location while creating a user profile on Twitter). In this work, we seek to propose a new framework, SpaceTimeWalk (*STWalk*), that seeks to utilize only the structural properties of time-varying graphs to learn compact low-dimensional embeddings capturing spatio-temporal properties of node trajectories.

We now present the proposed *STWalk* algorithm.

2.3 Network Representation Learning

2.3.1 Node Classification

In recent years, there have been a few approaches proposed for learning representations of networks, in particular, learning graph node embeddings. Many of the recent efforts have been inspired by the recent wave of deep learning methods that have demonstrated impressive success in learning representations of other kinds of data such as images, speech and text. DeepWalk [27], one of the earliest efforts in this regard, combined a random walk-based method with a SkipGram network (traditionally used in Natural Language Processing) to learn node representations. Node2vec [28] extended DeepWalk by employing biased random walks to learn node embeddings. In particular, Node2Vec precomputes a matrix, which is similar to a transition probability matrix, that assigns a probability of transition in a random walk, and is based on two hyper-parameters that control whether the walk will stay within a close neighborhood of the node or it will move farther away from that node.

Tang et al. (LINE) [29] computed node embeddings using a function of the probability of observing first-order and second-order nodes together in a random walk within a certain window size. This method then uses Asynchronous Stochastic Gradient Descent to solve the optimization problem. GraRep [30] employed a Singular Value Decomposition-based method for dimensionality reduction

and learning graph representations. Cao et al. [31] proposed a stacked denoising autoencoder for learning the vertex representations in graphs, where they used random surfer model to learn the structural properties of graph calculate the positive pointwise mutual information and then employ denoising autoencoder to reduce the dimensions of learned node features. There have also been efforts, such as [32], which have used deep neural network-based models for learning embeddings in heterogeneous networks. The main goal of the paper [32] was to consider information from multiple sources in dynamic network and learn unified representation. A more detailed review on network embedding techniques can be found in [33]. However, all the aforementioned methods work only with static graphs. In this work, we focus on learning representations of node trajectories in dynamic graphs. Dynamic graphs are evolving by nature, and it is important to capture the addition and deletion of nodes and edges, while capturing the behavior of a node over time in a learnt representation.
Node classification using Attention: Many algorithms have been proposed for node classification. However, recently researchers have started using attention for this task. The main aim of using the attention is to pick only few neighbors whose features are most helpful in discriminating the current node. This problem of focusing on important neighbors is solved by taking a weighted average of neighbors. The algorithms differ in the way they calculate the neighborhood weights. Masci et al [34] propose an algorithm for 3D shapes that works in non-Euclidean domain. The neighborhood weight is calculated based on the distance between two nodes on a polar coordinate system. Boscaini et al [35] use the heat kernel to measure the distance between two nodes and calculate the neighborhood weight. Monti et al [36] propose to learn mixture of Gaussian distributions on the neighborhood that will learn how to weight the neighbors instead of using predetermined weighing function. Velickovic et al [37] propose similar algorithm. Atwood et al [38] proposed a diffusion-convolution operation for node classification. Our work differs from these algorithms as we propose to use attention for graph classification i.e. selecting important nodes out of all the nodes in graph which are helpful in classifying the given graph. Our motivation for using attention is, the real-world graphs can be very huge and hence it is not feasible solution to use all nodes to classify the graph. Moreover, only certain area of graph could be playing important part in distinguishing the graph from others. Hence, in this work we propose SoftAttention module which picks the most important nodes in the graph that will help it classify the graph correctly.

2.3.2 Graph Classification

Early work in the domain of graph classification include kernel based algorithms such as Shortest-path kernel [39], random-walk kernel [40], graphlet count kernel [41] and WL kernel [42]. However, the main issue with these algorithms is they are not scalable and non-differentiable. In recent time, many deep learning based approaches have been proposed. Niepert et al [43] proposed a CNN based architecture for graph classification. The algorithm includes multiple preprocessing steps that coverts the given graph in such a way that CNN can be applied on it. Subgraph2vec [44] algorithm uses information from nearby nodes to learn the latent representation of rooted subgraph in unsupervised setting and these features are then used for graph classification. Deep Graph Kernel [45] count the occurrences of small size graphlets in given graph. It is used as feature vector for graph classification. Zhang et al [46] propose a pooling layer which sorts the nodes and considers first few nodes for graph classification. Our work is different than these algorithm as it is end-to-end differentiable, with SelfAttention module that selects the most important nodes in the graph, SelfOrdering module that orders the nodes in such a way that it can perform better at graph classification.

Chapter 3

Community-based Outlier Detection for Edge-Attributed Graphs

3.1 Motivation

Outlier (or anomaly) detection is a very broad field which has been studied in the context of a large number of research areas like statistics, data mining, sensor networks, environmental science, distributed systems, spatio-temporal mining, etc. Many algorithms have been proposed for outlier detection in high-dimensional data, uncertain data, stream data and time series data. By its inherent nature, network data provides very different challenges that need to be addressed in a special way. Network data is gigantic, contains nodes of different types, rich nodes with associated attribute data, noisy attribute data, noisy link data, and is dynamically evolving in multiple ways. Outlier detection on networks for various applications can be useful for: (1) identification of interesting entities or sub-graphs, (2) data de-noising (both with respect to the network nodes and edges), (3) understanding the anomalous temporal behavior of entities, and (4) identification of new trends or suspicious activities in both static and dynamic scenarios. Some examples of network outliers include users who spread rumors on Twitter, unusual but successful associations of persons with various organizations in an organizational network, sensor with anomalous readings in a sensor network, anomalous traffic between two components in a distributed network, snapshot with broken correlation between network properties across time, etc.

Given a static graph, one can find node, edge or subgraph outliers. For dynamic graphs, the outliers could be time stamps where the properties of the snapshot are significantly different from the properties of other snapshots. In the dynamic scenario, one can also identify a node (or an edge, or a subgraph) which shows anomalous behavior across time as an outlier. Popular outlier detection methods for static graphs include the Minimum Description Length (MDL) method [47], frequent subgraph mining, egonet analysis [48], and community analysis [9, 10]. Popular methods for outlier detection for dynamic graphs include graph similarity based methods [49], temporal spectral analysis [50], temporal structural connectivity analysis [51], and temporal community analysis [16, 15].

While most work in the past on network outlier detection has focused on graphs with node data and links, there is very little work in the area of edge-attributed graphs. In some cases, node data and linkage could be normal even when edge data is abnormal, in the context of nodes on which the edge is incident. With the rich variety of interactions in various complex graphs across a variety of domains, it is critical to incorporate such edge attributes in finding novel outliers. If we could assign a latent community to every node and every edge, a HCOutlier node is one which is linked to many nodes of another community, or has incident edges belonging to another community.

The following real-world examples demonstrate the importance of HCOutliers.

Organizational Graph Example: A graph of people working in a company is called an organization graph. Edges represent frequent communication. Role of a person can be considered as the community label. Usually a chemist in a company would communicate frequently with other chemists. But a suspicious HCOutlier node would not just communicate frequently with members of other roles, but also the community of the communications would be different from the one expected for its role. Such a person could be an activist planning a malicious activity with other folks in the company, or a star performer working on a secret collaborative project.

Co-authorship Graph Example: A co-authorship graph contains authors as nodes. Two authors are connected if they have published a paper together. On such graphs, research areas can be considered as latent communities. Most authors collaborate with other authors, and publish papers *in the same research area.* An HCOutlier would be an author who: (1) collaborates frequently with authors from other research areas, and (2) collaborates with other authors of same or different community on papers which belong to other research areas. Such authors perform inter-disciplinary research and are vital in cross-pollination of concepts across research areas.

Co-actorship Graph Example: A co-actorship graph contains actors as nodes. Two actors are connected if they have worked in a movie together. On such graphs, movie genres can be considered as latent communities. Most actors collaborate with other actors, and work in movies *in the same genre.* An HCOutlier would be an actor who: (1) collaborates frequently with actors from other genres, and (2) collaborates with other actors of same or different genre in movies which belong to other genres. HCOutlier actors are ones who have built a reputation for being multi-faceted actors with an eclectic range of movies.

Limitations of Past Approaches: In the past, various approaches have been proposed by looking only at node data and ignoring network structure. Clearly, HCOutliers cannot be detected using such a global approach because HCOutlier nodes belong to popular communities. There has been work on finding community outliers for both homogeneous [9] and heterogeneous graphs [10] by considering node data and link structure but ignoring edge data. Clearly, such an approach can identify some HCOutliers who are linked to nodes with a different community label. But they cannot identify a HCOutlier who could be connected to nodes of the same community label but has incident edges with a different community label.

Brief Overview of the Proposed Approach: We propose a probabilistic model for detection of Holistic Community Outliers. We model the problem as a community detection task where the outliers are modeled as a separate community. A Hidden Markov Random Field (HMRF) is used to perform joint community analysis for both nodes and edges considering node data, edge data and the linkage structure. Thus, the HMRF contains both nodes and edges from the graph as vertices, referred to as a node-vertex and an edge-vertex respectively. Community labels for outliers are sampled from a uniform distribution while normal vertices could be sampled from Gaussian or a multinomial distribution. EM is used for the inference, and to compute the hidden community label Z for every vertex.

We make the following contributions in this chapter:

- We look at the community outlier detection problem from a holistic perspective by incorporating node data, edge data, and the linkage structure. Based on such a view, we propose the problem of detecting novel HCOutliers.
- We model the problem as a community analysis task over a HMRF and provide inference using an EM algorithm.
- Using several experiments on synthetic datasets and a DBLP dataset, we show the effectiveness of the proposed approach in identifying HCOutliers.

3.2 Methodology

In this section, we develop the HMRF model which we use to perform joint community analysis for both nodes and edges considering node data, edge data and the linkage structure.

3.2.1 Problem Statement

Consider an undirected graph $G = \{V, E\}$, where V is the set of vertices and E is the set of edges. Node and Edge Data: Let S be the set of observed data.

- Let S_i represent the observed data of node $v_i \in V$ which has p attributes, S_{i_1}, S_{i_2}, $\cdots$, S_{i_p}.
- Let S_{ij} represent the observed data of edge $e_{ij} \in E$, i.e., edge data between nodes v_i and v_j, and there are q number of attributes, S_{ij_1}, S_{ij_2}, $\cdots$, S_{ij_q}.

HMRF Model: The HMRF contains a vertex representing each node and each edge from G. We refer to the HMRF vertices representing nodes and edges as node-vertices and edge-vertices respectively. We use the index i to refer to node-vertices in the HMRF, and the index ij to refer to edge-vertices in the HMRF. We use the index b to refer to all the vertices in the HMRF. Thus $b \in B = \{1, 2, \cdots, i, \cdots, |V|, (1,2), (1,3), \cdots, (1, |V|), (2,3), \cdots, (|V|-1, |V|)\}$. Note that $ij \in B$ if $i < j$. Thus, $|B| = |V| + |E|$.

- Let $X = \{X_b\}$ be a set of random variables. X_i generates node data S_i, and X_{ij} generates edge data S_{ij}.
- Let $Z = \{Z_b\}$ be the set of hidden random variables. Z_i indicates the community assignment of v_i. Suppose there are K communities, then $Z_i \in \{0, 1, 2, \cdots, K\}$. If $Z_i = 0$, then v_i is an outlier. If $Z_i = k$, then v_i belongs to the community k. Similarly, Z_{ij} denotes the community assignment for edge e_{ij}.

- Let W denote the weights in the HMRF. W_{v_i,v_j} is set to the weight of the edge $e_{ij} \in E$ and represent edge strength. $W_{v_i,v_j,e_{ij}}$ are weights for a triangle clique consisting of v_i, v_j and e_{ij}. Thus, $|W| = 4|E|$ (3 faces of the triangle formed by two node-vertices and an edge-vertex, and the whole triangle itself).

$W_{v_i,e_{ij}}$ can be set in multiple ways. We set it to the inverse of number of neighbors of v_i in G. Thus, for a co-authorship graph, this weight can be set to the inverse of the number of co-authors of author v_i in the graph. For a Twitter users graph, this weight can be set to the inverse of the number of other users that the user v_i is connected to. In a directed network sense, the weight can also be set to the ratio of the number of tweets from user v_i to user v_j to number of total tweets from user v_i.

$W_{v_i,v_j,e_{ij}}$ should reflect some property of the triangle clique which cannot be captured using individual edges. For a co-authorship network, this could be set to ratio of number of papers where authors v_i and v_j are co-authors to the total number of papers on which either v_i or v_j are authors. Similarly, for a Twitter user network, this weight can be set to the ratio of tweets exchanged between users v_i and v_j to the total tweets posted by either v_i or v_j.

Z_b's are influenced by their neighbors, i.e., if two nodes are linked, it is likely that both belong to the same community. This does not hold for the outliers. Let N_i denote the set of neighbors of node v_i. If $Z_i = 0$, then v_i is an outlier, and so the neighborhood set is empty. Also, each node-vertex is connected to the edge-vertex corresponding to the edges which are incident on the node in G.

$$N_i = \begin{cases} \{v_j | W_{v_i v_j} > 0, i \neq j, Z_j \neq 0\} \cup & \\ \{e_{xy} | W_{v_i e_{xy}} > 0, x = i \; or \; y = i, Z_{xy} \neq 0\} & Z_i \neq 0 \\ \phi & Z_i = 0 \end{cases} \tag{3.1}$$

Similarly, we also define neighbors for edge-vertices, N_{ij}, as follows. Note that for an edge-vertex, the neighbors are only the node-vertices on which the edge-vertex is incident except for the case when these node-vertices are outliers.

$$N_{ij} = \begin{cases} \{v_i | z_i \neq 0\} \cup \{v_j | z_j \neq 0\} & Z_{ij} \neq 0 \\ \phi & Z_{ij} = 0 \end{cases} \tag{3.2}$$

Data Likelihood: Let $\Theta = \{\theta_1, \theta_2, .., \theta_K\}$ be the set of parameters describing the normal communities. Thus, likelihood for node/edge data can be written as follows.

$$P(X_b = S_b | Z_b = k) = P(X_b = S_b | \theta_k) \tag{3.3}$$

We assume the outliers follow uniform distribution as we do not know beforehand which elements(node-vertex or edge-vertex) are anomalous or what they look like.

$$P(X_b = S_b | Z_b = k) = \rho_0 \tag{3.4}$$

where ρ_0 is a constant.

Figures 3.1 and 3.2 illustrate a sample graph and its corresponding HMRF model. The graph has two communities, blue and red. The node v_5 has connection with the member of blue community as if it is part of blue community but its node attributes are similar to red community. Hence, in the HMRF model, it is not connected to any other node, indicating v_5 as an outlier. The edge between v_4 and v_6 indicate the cross-community collaboration and thus detected as interesting nodes.

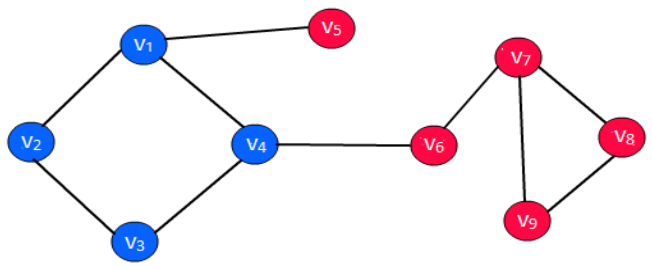


Figure 3.1: Original Graph G=(V,E)

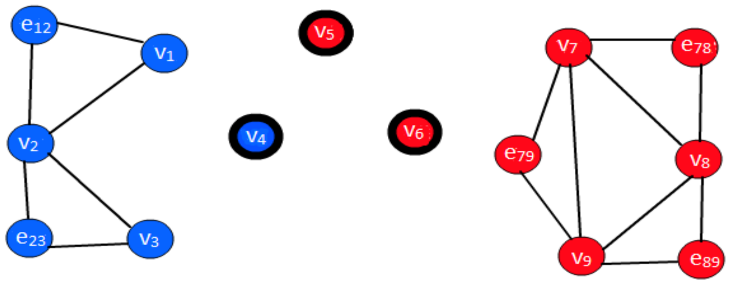


Figure 3.2: HMRF model of G indicating anomalous node v_4, v_5 and v_6

3.2.2 Cliques and Potentials in HMRF

As the community label assignment follows an HMRF, we can define $P(Z) = \frac{1}{H_1}\exp(-U(Z))$ where $U(Z)$ is the potential function, defined as sum over all possible cliques.

We consider two kinds of clique potentials: pairwise (v_i to v_j, v_i to e_{ij}) and triangular (between v_i, v_j and e_{ij}). Then, the potential function can be written as follows.

$$\begin{aligned}U(Z) = -\lambda_1 &\sum_{\substack{W_{v_iv_j}>0,\\ Z_i\neq 0,\\ Z_j\neq 0}} W_{v_iv_j}\delta(Z_i - Z_j)\\ -\lambda_2 &\sum_{\substack{W_{v_ie_{ij}}>0,\\ Z_i\neq 0,\\ Z_{ij}\neq 0}} W_{v_ie_{ij}}\delta(Z_i - Z_{ij})\\ -\lambda_2 &\sum_{\substack{W_{v_je_{ij}}>0,\\ Z_j\neq 0,\\ Z_{ij}\neq 0}} W_{v_je_{ij}}\delta(Z_j - Z_{ij})\\ -\lambda_3 &\sum_{\substack{W_{v_iv_je_{ij}}>0,\\ Z_i\neq 0,\\ Z_j\neq 0,\\ Z_{ij}\neq 0}} W_{v_iv_je_{ij}}\psi(Z_i, Z_j, Z_{ij})\end{aligned} \tag{3.5}$$

where $\lambda_1, \lambda_2, \lambda_3$ are constants, $W_{v_iv_j} > 0$, $W_{v_ie_{ij}} > 0$, $W_{v_je_{ij}} > 0$, $W_{v_iv_je_{ij}} > 0$, imply that there are links connecting v_i-v_j, v_i-e_{ij}, v_j-e_{ij} and Z_i, Z_j, Z_{ij} are non-zero. The δ function is defined as $\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ otherwise. Also, $\psi(Z_i, Z_j, Z_{ij}) = 1$ if $\delta(Z_i - Z_j) + \delta(Z_i - Z_{ij}) + \delta(Z_j - Z_{ij}) \leq 1$, and 0 otherwise. Note that we define ψ this way to capture the intuition that the clique potential should fire if at least two of the members in the clique share the same community label. Overall, the potential function suggests that, if members of the clique are normal objects, they are more likely to be in the same community when there exists a link connecting them in G, and the probability becomes higher if their link (weight) is stronger.

3.2.3 Probability of Generating the Data

Given the community label of a vertex (Z_i or Z_{ij}) in the HMRF, the corresponding data (X_i or X_{ij}) can be modeled as data generated from a distribution with parameters specific to the community. Community labels can be obtained using either Gaussian mixture model on the data if it is continuous, or as multinomial if the data is textual.

Continuous Data: If the data is numeric and 1-dimensional, we need to learn the mean (μ_k), and standard deviation (σ_k) for each community k represented using a Gaussian. Given the parameters and the community label, the log likelihood can be written as follows.

$$\ln P(X_i = S_i | Z_i = k) = -\frac{(S_i - \mu_k)^2}{2\sigma_k^2} - \ln \sigma_k - \ln \sqrt{2\pi} \tag{3.6}$$

In case of multi-dimensional data with p dimensions, the log likelihood can be written as follows.

$$\begin{aligned}\ln P(X_i = S_i | Z_i = k) = &-\frac{(S_i - \mu_k)^T \Sigma_k^{-1} (S_i - \mu_k)}{2}\\ &-\frac{\ln \Sigma_k}{2} - \ln \sqrt{(2\pi)^p}\end{aligned} \tag{3.7}$$

Discrete Data: If the node (or edge) data can be represented as a document, we assume that the attributes (or words) are independent of each other, given the community label.

$$P(X_i = S_i|\theta_k) = \prod_{c=1}^{p} P(X_{i_c} = S_{i_c}|\theta_k) \tag{3.8}$$

$$P(X_{ij} = S_{ij}|\theta_k) = \prod_{c=1}^{q} P(X_{ij_c} = S_{ij_c}|\theta_k) \tag{3.9}$$

For text data, given a vocabulary of T words, let d_{bl} be the number of times the word l appears in data related to vertex b. Then the parameters $\theta_k = \{\beta_{k1}, \cdots, \beta_{kT}\}$ denote the probability of a particular word belonging to community k. Given the parameters, the data likelihood of an object belonging to the k^{th} community can be computed as follows.

$$\ln P(X_{i_c} = S_{i_c}|Z_i = k) = \sum_{l=1}^{T} d_{il} \ln \beta_{kl} \tag{3.10}$$

where β_{kl} is the probability of the word with index l belonging to community k.

3.3 Inference

The model parameters Θ and the set of hidden labels Z are unknown as described in Section 3.2. In this section, we describe methods to infer the hidden labels and also estimate the model parameters.

3.3.1 Inferring Hidden Labels

Assuming the model parameters Θ are known, we find the configuration Z that will maximize the posterior distribution as follows.

$$\hat{Z} = \arg\max_{Z} P(X = S|Z)P(Z) \tag{3.11}$$

To find such a configuration, we use the Iterated Conditional Modes (ICM) algorithm [52]. It is a greedy algorithm which guarantees convergence to a local optima by sequentially updating one Z_b at a time assuming other Z's (i.e., Z_{B-b}) as constant. At each step, the algorithm updates Z_b given $X_b = S_b$ and the other labels such that the posterior $P(Z_b|X_b = S_b, Z_{B-b})$ is maximized. If $Z_b \neq 0$, applying Bayes rule,

$$P(Z_b|X_b = S_b, Z_{B-b}) \propto P(X_b = S_b|Z)P(Z) \tag{3.12}$$

In Eq. 3.12, $P(Z)$ can be computed using Eq. 3.5. But rather than considering potential for the entire graph, Eq. 3.12 needs potentials defined over only those cliques in which Z_b is involved. Thus, we can write the following equation for node-vertices:

$$
\begin{aligned}
P(Z_i|X_i = S_i, Z_{B-\{i\}}) \propto P(X_i = S_i|Z)\times \\
\exp(\lambda_1 \sum_{v_j\in N_i} W_{v_i v_j}\delta(Z_i - Z_j) \\
+ \lambda_2 \sum_{e_{xy}\in N_i} W_{v_i e_{xy}}\delta(Z_i - Z_{xy}) \\
+ \lambda_3 \sum_{\substack{v_j\in N_i,\\ e_{xy}\in N_i}} W_{v_i v_j e_{xy}}\psi(Z_i, Z_j, Z_{xy}))
\end{aligned}
\tag{3.13}
$$

Also, the corresponding equation for the edge-vertices can be written as follows.

$$
\begin{aligned}
P(Z_{ij}|X_{ij} = S_{ij}, Z_{B-\{(i,j)\}}) \propto P(X_{ij} = S_{ij}|Z)\times \\
\exp(\lambda_2 \sum_{v_{i'}\in N_{ij}} W_{v_{i'} e_{ij}}\delta(Z_{i'} - Z_{ij}) \\
+ \lambda_3 W_{v_i v_j e_{ij}}\psi(Z_i, Z_j, Z_{ij}))
\end{aligned}
\tag{3.14}
$$

Taking log of the posterior probability, we can transform the maximizing posterior problem to minimizing the conditional posterior energy function defined as shown in Eqs. 3.15 and 3.16 for node-vertices and edge-vertices respectively.

$$
\begin{aligned}
U_i(k) = -\ln P(X_i = S_i|Z_i = k) - \lambda_1 \sum_{v_j\in N_i} W_{v_i v_j}\delta(k - Z_j) \\
- \lambda_2 \sum_{e_{xy}\in N_i} W_{v_i e_{xy}}\delta(k - Z_{xy}) \\
- \lambda_3 \sum_{\substack{v_j\in N_i,\\ e_{xy}\in N_i}} W_{v_i v_j e_{xy}}\psi(k, Z_j, Z_{xy})
\end{aligned}
\tag{3.15}
$$

$$
\begin{aligned}
U_{ij}(k) = -\ln P(X_{ij} = S_{ij}|Z_{ij} = k) \\
- \lambda_2 \sum_{v_{i'}\in N_{ij}} W_{v_{i'} e_{ij}}\delta(Z_{i'} - k) - \lambda_3 W_{v_i v_j e_{ij}}\psi(Z_i, Z_j, k)
\end{aligned}
\tag{3.16}
$$

If $Z_b = 0$, the vertex has no neighbors, and thus

$$
P(Z_b|X_b = S_b, Z_{B-\{b\}}) \;\propto\; P(X_b = S_b|Z_b = 0)P(Z_b = 0) \tag{3.17}
$$

Hence,

$$
U_b(0) = -\ln(\rho_0\pi_0) = a_0 \tag{3.18}
$$

Finally, the label Z_b for vertex b in HMRF is set to $k \in \{0, 1, \cdots, K\}$ such that $U_b(k)$ is minimized. The predefined hyper-parameters, $\lambda_1, \lambda_2, \lambda_3$, represent the importance of the respective components of the graph structure. a_0 is the outlierness threshold. Algorithm 8 first randomly initializes the labels for all vertices. At each step, labels are updated sequentially by minimizing $U_b(k)$.

Algorithm 1: Inferring Hidden Labels

```
1 Input: Node/Edge data $S$, weights $W$, set of model parameters $\theta$, number of clusters $K$,
    hyper-parameters $(\lambda_1, \lambda_2, \lambda_3)$, threshold $a_0$, initial assignment of labels $Z^{(1)}$;
2 Output: Updated assignment of labels $Z$;
3 Randomly set $Z^{(0)}$ $t \leftarrow 1$ while $Z^{(t)}$ is not close enough to $Z^{(t-1)}$ do
4     $t \leftarrow t + 1$ for $b = 1; b \leq |B|; i++$ do
5         Update $Z_b^{(t)} = k$ which minimizes $U_b(k)$ using Eqs. 3.15, 3.16 and 3.18
6     end
7 end
8 return $Z^{(t)}$
```

3.3.2 Estimating Parameters

In this subsection, we discuss a method for estimating unknown θ from the data. θ describes the model that generates node data and edge data. Hence we seek to maximize the data likelihood $P(X = S|\theta)$ to obtain $\hat{\theta}$. However, since both the hidden labels and the parameters are unknown and they are inter-dependent, we use expectation-maximization (EM) algorithm (as shown in Algorithm 6) to solve the problem. We direct the reader to [9] for details.

The algorithm starts with an initial estimate of hidden labels $Z^{(1)}$. Given the current configuration of hidden labels, we can estimate model parameters as follows. For univariate numeric data, $\mu_k^{(t+1)}$ and $\sigma_k^{(t+1)}$ are estimated as follows:

$$\mu_k^{(t+1)} = \frac{\sum_{b=1}^{|B|} P(Z_b = k|X_b = S_b, \Theta^{(t)}) S_b}{\sum_{b=1}^{|B|} P(Z_b = k|X_b = S_b, \Theta^{(t)})} \tag{3.19}$$

$$(\sigma_k^{(t+1)})^2 = \frac{\sum_{b=1}^{|B|} P(Z_b = k|X_b = S_b, \Theta^{(t)})(S_b - \mu_k)^2}{\sum_{b=1}^{|B|} P(Z_b = k|X_b = S_b, \Theta^{(t)})} \tag{3.20}$$

For text data, given a vocabulary of T words, let d_{bl} be the number of times the word l appears in data related to vertex b. Then, given the current configuration of hidden labels, we can estimate parameters of the multinomial distribution β_{kl} for $k \in \{1, \cdots, K\}$ and $l \in \{1, \cdots, T\}$ as follows:

$$\beta_{kl}^{(t+1)} = \frac{\sum_{b=1}^{|B|} P(Z_b = k|X_b = S_b, \Theta^{(t)}) d_{bl}}{\sum_{l=1}^{T} \sum_{b=1}^{|B|} P(Z_b = k|X_b = S_b, \Theta^{(t)}) d_{bl}} \tag{3.21}$$

Given the updated parameters, the new configuration of hidden labels can be estimated using Algorithm 8 where $P(Z_b = k^*|X_b = S_b, \Theta^{(t)}) = 1$ if $k^* = \arg\min_k U_b(k)$, and 0 otherwise.

In summary, the Holistic Community Outlier detection algorithm shown in Algorithm 6 works as follows. It begins with some initial label assignment of the vertices in the HMRF. In the M-step, the model parameters are estimated using the EM algorithm to maximize the data likelihood, based on the current label assignment. In the E-step, Algorithm 8 is run to re-assign the labels to the objects by minimizing $U_b^{(k)}$ for each HMRF vertex sequentially. The E-step and M-step are repeated until convergence is achieved, and the outliers are the nodes that have 0 as the final estimated labels.

Algorithm 2: Holistic Community Outlier Detection

```
1 Input: Node/Edge data $S$, weights $W$, set of model parameters $\theta$, number of clusters $K$,
    hyper-parameters $(\lambda_1, \lambda_2, \lambda_3)$, threshold $a_0$, initial assignment of labels $Z^{(1)}$;
2 Output: Set of Holistic Community Outliers;
3 Randomly set $Z^{(0)}$ $t \leftarrow 1$ while $Z^{(t)}$ is not close enough to $Z^{(t-1)}$ do
4     M-step: Given $Z^{(t)}$, update parameters $\Theta^{(t+1)}$ according to Eqs. 3.19, 3.20 or 3.21.
       E-step: Given $\Theta^{(t+1)}$, update the hidden labels as $Z^{(t+1)}$ using Algorithm 8. $t \leftarrow t + 1$
5 end
6 return the indices of outliers: $\{i : Z_b^{(t)} = 0, b \in B\}$
```

3.4 Discussion

In this section, we discuss how to set the hyperparameters and how to initialize the hidden labels.

Setting Hyperparameters: The HCOutlier detection algorithm has the following hyperparameters: threshold a_0, clique importance variables, $\lambda_1, \lambda_2, \lambda_3$ and number of communities K.

$\lambda_1, \lambda_2, \lambda_3$ indicate the importance of link between two nodes, the importance of link between edge-vertex and a node-vertex, and the importance of the triangle between the edge-vertex and the node-vertices of which the edge is incident respectively. If λs are set to low values, the algorithm will consider only node information for finding the outliers. If their values are set too high, all connected nodes will have the same label, so an upper bound can be set for $\lambda_1 + \lambda_2 + \lambda_3$, such that a value above this bound will result into empty communities. High λ_1 will give more importance to the linkage in the graph. High λ_2 will give high importance to consistency between the node data and edge data in the graph. High λ_3 will give high importance to consistency between the edge data and both the incident nodes in the graph.

The threshold a_0 can be replaced by another parameter (percentage of outliers r) as follows. In Algorithm 8, first calculate $\hat{Z}_i = \arg\min_k U_i(k)(k \neq 0)$ for each $i \in \{1, \cdots, |V|\}$ and then sort $U_i(\hat{Z}_i)$ in non-descending order. Finally, select top r percentage as outliers.

K represents the number of normal communities. For small value of K, algorithm will find the global outliers, and for large value of K many local outliers will be detected because of many communities. An appropriate K can be set using a variety of methods like Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), Minimum Description Length (MDL), etc.

Initialization of labels: Instead of initializing Z randomly, we initialize Z values by clustering the nodes without considering outliers. To overcome local optima issues, we run the algorithm multiple times with different initialization, and choose the one with the largest data likelihood value.

3.5 Experiments

Evaluation of outlier detection algorithms is difficult in general due to lack of ground truth. Hence, we perform experiments on multiple synthetic datasets. We evaluate outlier detection accuracy of the proposed algorithm based on outliers injected in synthetic datasets. We evaluate the results on real datasets using case studies. We perform comprehensive analysis of objects to justify the top

Table 3.1: Synthetic Dataset Results on Graphs A, B and C (K=5; OD Acc.=Outlier Detection Accuracy, CA Acc.=Community Assignment Accuracy; CODA is Baseline, HCODA is the proposed algorithm)

$\|V\|$	% injected	% extracted	CODA [9] OD Acc.	HCODA OD Acc.	CODA [9] CA Acc.	HCODA CA Acc.	Average runtime of HCODA (ms)
1000	1	1	1.000	1.000	0.714	0.713	898.6
1000	5	1	1.000	1.000	0.725	0.735	930
1000	5	5	0.760	0.800	0.739	0.756	899
1000	10	1	1.000	1.000	0.683	0.699	917.2
1000	10	5	0.960	0.980	0.720	0.738	913
1000	10	10	0.760	0.840	0.729	0.762	926
10000	1	1	0.690	0.730	0.770	0.784	9846.6
10000	5	1	0.970	1.000	0.740	0.755	10259
10000	5	5	0.746	0.780	0.758	0.777	10478.8
10000	10	1	1.000	1.000	0.705	0.718	10026.8
10000	10	5	0.958	1.580	0.741	0.757	9840.4
10000	10	10	0.795	0.814	0.759	0.777	9626.8
100000	1	1	0.722	0.735	0.773	0.787	120587
100000	5	1	0.995	0.996	0.747	0.762	113171.5
100000	5	5	0.751	0.789	0.764	0.783	113749.2
100000	10	1	0.997	0.999	0.698	0.714	104812.4
100000	10	5	0.946	0.974	0.733	0.751	103004
100000	10	10	0.784	0.808	0.750	0.770	100314

few outliers returned by the proposed algorithm. The code and the data sets are publicly available[1]. All experiments are performed on a machine with Intel core i3-2330M processor running at 2.20GHz and 4GB RAM.

We compare our algorithm with Community Outlier Detection Algorithm (CODA) [9]. CODA is also an algorithm for community-based outlier detection which performs community detection using node data and linkage; but ignores edge-data completely.

3.5.1 Synthetic Dataset

We synthetically generate three graphs: Graph A ($|V|$=1000), Graph B ($|V|$=10000) and Graph C ($|V|$=100000). The node data is generated using 5 different Gaussian distributions and similar procedure is followed to generate edge data as well. Then for each of these graphs, different percentage of outliers are injected by randomly changing the data associated with 1%, 5% and 10% of the nodes.

We generate a variety of synthetic datasets by varying the number of nodes in the graph, percentage of outliers injected in the graph and percentage of outliers to be extracted by the algorithm. Table 3.1 shows the synthetic dataset results for the baseline (CODA) as well as the proposed algorithm (HCODA). We set the number of clusters (K) to 5 for these experiments (based on the generation process). We measure the performance of the two algorithms from two perspectives: firstly, accuracy of extraction of the injected outliers (OD Acc.) and secondly, how well it assigns the community label to all the nodes of the graph (CA Acc.). Note that each cell of the table corresponds to average over five runs of the algorithm. As the table shows, the proposed algorithm (HCODA) is 6.2% more accurate than the baseline on average in terms of outlier detection accuracy. Similarly, HCODA is 2.2% better than CODA on average. The performance is consistent across various graph sizes as well as across different degrees of outlierness in the graph.

[1]https://github.com/supriya-pandhre/HCODA

Hyperparameter Sensitivity: To understand the impact of various hyperparameters (λ's), we performed a few experiments. The results are shown in Figures 3.3, 3.4 and 3.5. Figure 3.3 shows the sensitivity of the outlier detection accuracy with respect to variation of λ_3. We plot the curves for Graph A and B for the case of 10% injected outliers and 10% outliers to be extracted. Recall that λ_3 is the weight for the triangular clique. As can be seen from the figure, higher values of λ_3 are preferable but in general any value greater than 0.5 is reasonable.

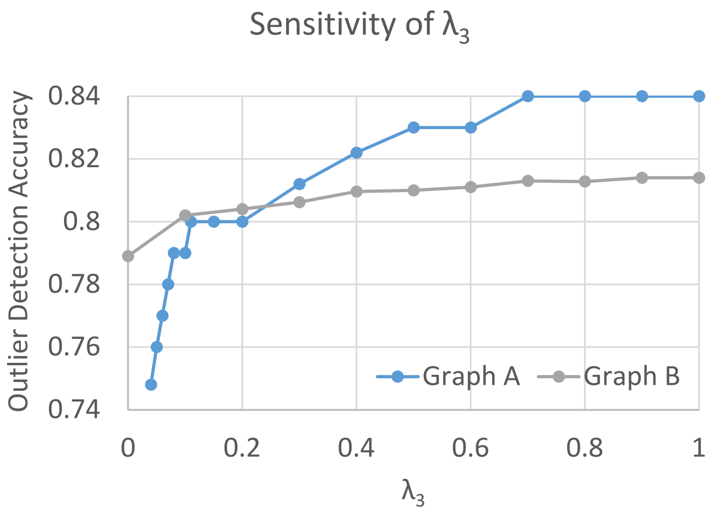


Figure 3.3: Sensitivity of λ_3

Figures 3.4 and 3.5 show the sensitivity of the outlier detection accuracy with respect to variation of λ_1 for both the proposed algorithm (HCODA) and the baseline (CODA). Figures 3.4 is the plot for Graph A while Figures 3.5 is the one for Graph B. Both the plots are for the case of 10% injected outliers and 1% outliers to be extracted. Recall that λ_1 is the weight for the linkage in the original graph. As can be seen from the figure, lower values of λ_1 are preferable both in the case of CODA as well as HCODA.

We noticed that the algorithm is not significantly sensitive to the parameter λ_2 and hence we do not show the corresponding plot.

3.5.2 Four Area Dataset

DBLP (`http://dblp.uni-trier.de/`) contains information about computer science journals and proceedings. Four Area is a subset of DBLP for the four areas of data mining (DM), databases (DB), information retrieval (IR) and machine learning (ML). It consists of papers from 20 conferences (five per area): KDD, PAKDD, ICDM, PKDD, SDM, ICDE, VLDB, SIGMOD, PODS, EDBT, SIGIR, WWW, ECIR, WSDM, IJCAI, AAAI, ICML, ECML, CVPR, CIKM. Thus, we consider the authors who have published papers in these conferences, and work with the co-authorship network (edge weight = co-authorship frequency). The edge data consisted of a vector of size four representing the count of papers co-authored in a particular research area.

The graph contains a total of 42844 author nodes. Each author is associated with a vector of size 20 containing the count of papers published by an author in the twenty conferences. There is an

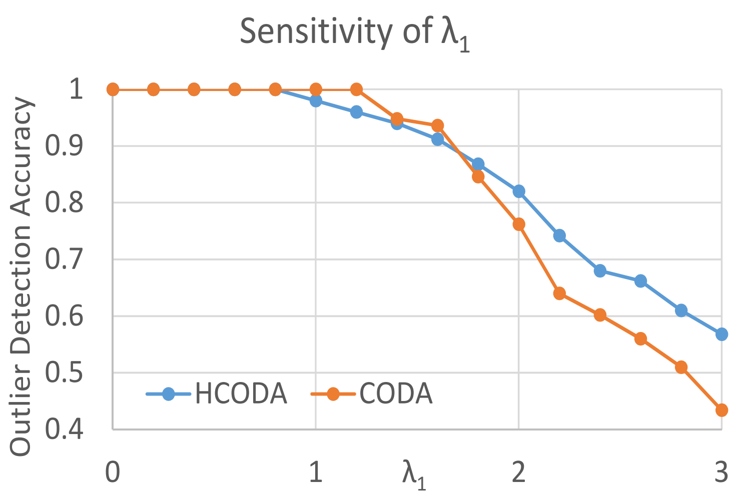


Figure 3.4: Sensitivity of λ_1 (Graph A)

edge between two authors if they co-authored a paper together and the count of such co-authored papers is the weight $W_{v_i v_j}$ between two nodes.

The graph contains 118618 co-authorship edges. The weight of the link between the author-vertex and the co-authorship-edge-vertex in the HMRF (i.e., $W_{v_i e_{ij}}$) is set to the inverse of the number of co-authors of author v_i in the graph. $W_{v_i, v_j, e_{ij}}$ is set to ratio of number of papers where authors v_i and v_j are co-authors to the total number of papers on which either v_i or v_j are authors. We set the number of communities to K=4 (i.e., four normal communities and an outlier community). Also, we set the percentage of outlier parameter r as 1%. The average execution time of the lgorithm is 30.32 seconds.

Case Studies: Result of algorithm show that it is able to identify interesting nodes(authors) and edges(co-authorship). We validate the result produced by algorithm by manually visiting the authors' homepages. For example, work by Sameer K. Antani is mainly in Clinical data standards and electronic medical records, but he published a paper with authors working in information retrieval i.e. node data of Sameer K. Antani is different from the fellow co-authors but the relationship is similar to what two authors from information retrieval should have and hence the algorithm has identified this node as interesting one. Similarly, Ivo Krka mainly focuses on software engineering and system modeling, however, he has published a paper on data mining and hence identified by the algorithm.

The proposed algorithm also identified authors based on the edge information. We discuss a few cases now. Anindya Datta usually publishes in information systems, and Debra E. VanderMeer usually publishes work related to design science research, but their co-authored work is published on data management and very large databases topics. Sabyasachi Saha has co-authored papers with Sandip Sen on topics related to artificial intelligence and he has also published work on information and knowledge management with Pang-Ning Tan. The algorithm was able to identify him because of his work in multiple research areas.

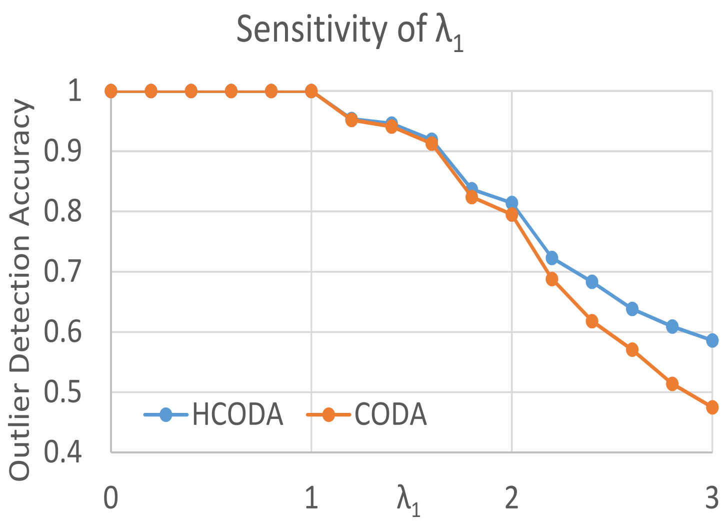


Figure 3.5: Sensitivity of λ_1 (Graph B)

3.6 Conclusion

In this chapter, we introduced a method (HCODA) that detects novel Holistic Community Outlier graph nodes by taking into account the node data and edge data simultaneously to detect anomalies. We modeled the problem as a community detection task using a Hidden Random Markov Model, where outliers form a separate community. We used ICM and EM to infer the hidden community labels and model parameters iteratively. Experimental results show that the proposed algorithm consistently outperforms the baseline CODA method on synthetic data, and also identifies meaningful community outliers from the Four Area network data.

Chapter 4

STWalk: Learning Trajectory Representations in Temporal Graphs

4.1 Motivation

The rising use of social networks and various other sensor networks in real-world applications ranging from politics to healthcare has made computational analysis of graphs a very important area of research today. The use of machine learning for various graph analysis tasks such as community detection [53], link analysis [54], node classification [55] and anomaly detection [56] has exponentially increased over the last few years, considering the direct impact of the success of these methods on business outcomes. A wide variety of methods have been proposed in the aforementioned areas over the last few years. However, a large part of the efforts so far has focused on static graphs. Platforms such as Facebook, Twitter and LinkedIn correspond to graphs that change on a daily basis; however, the algorithms that analyze such graphs are primarily static, i.e., they consider a graph at a given snapshot of time for analysis. In this work, we seek to focus on temporal graphs, and on learning representations of node trajectories in such graphs.

Study of user-user interactions over time plays an important role in many applications, including user classification for targeted advertising or link prediction to suggest new connections on social networking sites. One of the crucial part of such applications is analyzing change in users' behavior over time. Aggarwal and Subbian [26] present a survey of methods that have been proposed so far to study temporal graphs, and categorize such methods into two kinds: *maintenance methods* that focus on adapting older models to a newer version of the same graph, or *analytical evolution analysis*, where the objective is to analyze the network as it changes itself. We propose to address the latter in this work. Recent efforts for analysis of temporal graphs have largely extended earlier methods proposed on static graphs, such as spectral methods or those that study statistical properties. In this work, we build on the recent emphasis on learning appropriate representations of data to temporal graphs. In particular, we propose SpaceTimeWalk (*STWalk*), which learns representations of node trajectories in temporal graphs. To the best of our knowledge, this is the first such effort.

STWalk is an unsupervised trajectory learning algorithm that embeds the spatial and temporal characteristics of nodes over a given time window. The algorithm carries out a random walk in a current graph, resulting in a *space-walk*, as well as in graphs at different time steps, called *time-walk*. The traversed paths are then considered as sentences, with nodes as words from a vocabulary. The SkipGram [1] network is then employed to learn the latent representations of node trajectories such that it maximizes the probability of co-occurrence of two nodes within the specified window size. We perform extensive experimentation on three real-world time-varying networks. Performance on these datasets show the effectiveness of our framework in dealing with different kinds of temporal networks. We also study the usefulness of these representations through visualization, as well as complementary tasks such as change point detection.

The remainder of this section is organized as follows. Section 4.2 presents the mathematical formulation of the problem, and the proposed methods are presented in Sections 4.2.1 and 4.2.2. The experimental results that study the performance of the proposed methods are described in Section 5.4, and Section 4.4 concludes the work with pointers to future directions.

4.2 Methodology

Learning the change in behavior of nodes in a temporal graph requires us to consider the information from the graph in the current time step, as well as graphs from previous time steps. Using the naïve way of examining graphs at each time-step separately will fail to capture the correlation information that exists between two graphs at consecutive time steps. Hence, we propose STWalk a random walk based framework that learns rich trajectory representations by capturing changes in node behavior across a given time window. We hence define the following to formulate our solution.

- Given a graph $G = (V, E)$ at T different time steps $\{G_1, G_2, G_3, \cdots, G_T\}$ having N number of nodes at each time step and varying number of edges $E_t, t \in \{1, \cdots, T\}$, where E_t represents the edges in G_t.
- Let W denote the adjacency tensor of size $N \times N \times T$, representing the adjacency matrix of the graph at different time stamps.

Our goal is to learn representations Φ that maps a given node to a d-dimensional representation. Let $N_t(u) \subset V$ denote the set of neighbors of node u in graph G_t. For a node u in graph G_t, the representation $\Phi_t(u)$ will be learned in such a way that, it maximizes the probability of observing the neighbors of u in graph G_t, viz. $N_t(u)$, and also the representation of u at previous time stamps $\Phi_{t-\tau}(u)$ where $\tau \in \{1, \cdots, t-1\}$. Equation (4.1) below formalizes this construction. Our objective hence is to obtain a d-dimensional representation for a node u at time t that maximizes the following log probability:

$$\max_{\phi_t} \sum_{u \in V} \log \Pr\left(N_t(u),\ \phi_{t-\tau}(u) \mid \phi_t(u)\right) \tag{4.1}$$

$$\text{where } \tau \in \{1, \cdots, t-1\}$$

We assume that the neighborhood nodes in $N_t(u)$ and past representations $\phi_{t-\tau}(u)$ are independent of each other, conditioned on the current representation of u, $\phi_t(u)$. Hence, we write:

$$\begin{aligned}&\log\left(\Pr(N_t(u), \phi_{t-\tau}(u)|\phi_t(u))\right)\\&= \log\left(\prod_{n_i \in N_t(u)} \Pr(n_i|\phi_t(u))\right) + \log\left(\Pr(\phi_{t-\tau}(u)|\phi_t(u))\right)\end{aligned} \tag{4.2}$$

Learning representations using random walk has proved to measure better graph proximity, and thereby improving the performance [33] [57]. Hence, we use random walk to learn the conditional probability of observing a node n_i given the learned representation $\phi_t(u)$ defined as follows:

$$\Pr(n_i|\phi_t(u)) = \frac{\exp(\phi_t(n_i)\phi_t(u))}{\sum\limits_{v \in N_t(u)} \exp(\phi_t(v)\phi_t(u))} \tag{4.3}$$

Therefore, to learn the trajectory representation, we maximize:

$$\begin{aligned}\max_{\phi_t} \sum_{u \in V} \log &\left(\prod_{n_i \in N_t(u)} \frac{\exp(\phi_t(n_i)\phi_t(u))}{\sum\limits_{v \in N_t(u)} \exp(\phi_t(v)\phi_t(u))}\right)\\ &+ \log\left(\Pr(\phi_{t-\tau}(u)|\phi_t(u))\right)\end{aligned} \tag{4.4}$$

While the above problem appears to be learning only the representation of a node at time t, it actually represents a representation of a node t considering its history over the graphs at the previous time steps, and hence, the term *node trajectory representation* in this work.

To solve the maximization problem defined in Equation 4.4, we propose two approaches. The first approach solves equation 4.4 by considering it as one single expression and learning a trajectory representation by maximizing the co-occurrence of neighborhood nodes at time t and in previous $t-\tau$ time-steps together. In the second approach, we learn the trajectory representation by solving two sub-problems separately: (i) maximizing the co-occurrence probability of a node and its neighbors at time t, and (ii) maximizing the co-occurrence probability of a node and its neighbors at preceding $t-\tau$ time-steps. The representations learned using these steps are then combined to form the final trajectory representation. Sections 4.2.1 and 4.2.2 describe each of the proposed approaches.

4.2.1 STWalk1

We describe the first approach, which we call STWalk1, in this section. For learning the trajectory representations of nodes in temporal graphs, here, we take into account the neighbors of a node at time steps t, $t-1$,..$t-\tau$ together and maximize the probability of observing these nodes together.

In STWalk1, to consider the neighbors of a node u from the current graph and graphs at previous time steps, we construct a Space-Time Graph. The space-time graph contains all nodes from graph at time t and it has a special temporal edge that links the node to itself in the graphs at previous time steps. Figure 4.1 illustrates this with an example of a space-time graph considered while learning the trajectory representation for node at time t, u_t. In particular, the figure shows the temporal edge (shown as a bold line) between u_t and past self-nodes u_{t-1}, u_{t-2} and u_{t-3}. The trajectory representation thus learned for u_t will hence be influenced by all the nodes in the Space-Time graph, as shown in Figure 4.1, viz. nodes from the present graph as well as its preceding self nodes and their

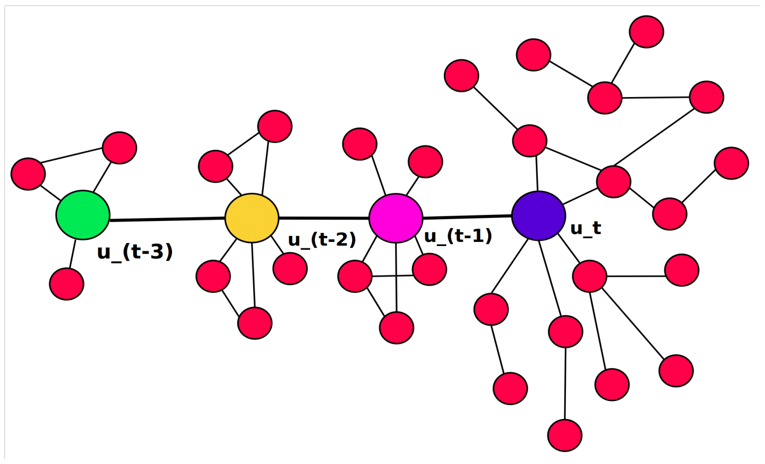


Figure 4.1: Example graph of node u_t generated by Algorithm createSpaceTimeEmb 3. The node u_t represents node in current graph and nodes $u\_(t-1)$, $u\_(t-2)$ and $u\_(t-3)$ indicate its past self-nodes. To learn trajectory representation for u_t, STWalk1 algorithm considers the influence of nodes in current time step as well as effect from the past.

first-level neighbors. Given a node and the window size, over which we want to learn the trajectory, Algorithm 3 describes the steps to create a Space-Time Graph starting from the given node.

Now, to learn the representations of nodes in the Space-Time graph, we perform random walks on it. If two nodes share many edges or neighbors, they will be visited more often in many random walks, indicating that these two nodes share a similar graph structure. Hence, the representation learned for these two nodes should be close to each other in the embedding space. This is analogous to the concept from Natural Language Processing (NLP) that if two words co-occur in many sentences, this indicates that they represent a similar context and hence their word vectors are nearby each other in word embedding space. To learn such a embedding that captures the relationship of a word with other co-occurring words in a window, Mikolov et al. [1] presented the SkipGram network. We use a similar SkipGram algorithm (given in Algorithm 4) to learn the node embeddings from random walks of a graph, similar to DeepWalk [27]. For more details of the SkipGram network, we request the interested reader to refer to [1].

For each node in the Space-Time graph, we perform ρ number of random walks each of length $\mathcal{L}$. The problem of learning the trajectory representation in such a way that it maximizes probability of co-occurrence of present and past nodes is reduced to maximizing the co-occurrence of nodes in a random walk within a vocabulary window $\mathcal{W}_v$. Hence, we use the SkipGram algorithm 4 to learn the trajectory representations for nodes. (In particular, we use the SkipGram network, as proposed in [1] to implement the SkipGram algorithm.) For each node in a graph G_t, two representations will be learned. One representation corresponds to the node embedding when it occurs in the context of other nodes, and the other representation which is learned for the node itself. The latter one is used as the node trajectory representation. The overall algorithm STWalk1 is illustrated in Algorithm 5.

4.2.2 STWalk2

In this section, we detail the second approach for learning the trajectory representation of nodes in a temporal graph. We solve the problem of learning representations of node trajectories, such

Algorithm 3: createSpaceTimeGraph

```
  Input: Set of graphs $\mathcal{G}$:$\{G_t, G_{t-1}, \cdots, G_{t-\tau}\}$, Time window size $\tau$, starting node $startNode$
         at time $t$
  Output: $G_{spacetime}$
1 $G_{spacetime} = G_t$
2 for each time step $i \in \{0, \cdots, \tau\}$ do
3     $pastNode = startNode$ at $G_{t-i}$
4     Create $pastNodeSubGraph$= the subgraph of $pastNode$ and its direct neighbors from
       $G_{t-i}$
5     Merge $G_{spacetime}$ and $pastNodeSubGraph$ to obtain updated $G_{spacetime}$
6     $startNode = pastNode$
7 end
```

Algorithm 4: SkipGram

```
  Input: Input representation: $\Phi$, List of nodes in a random walk: $walk$, windowSize: $\mathcal{W}_v$
  Output: Updated representation: $\Phi$
1 for $n \in walk$ do
2     for $n_i \in walk[n - \mathcal{W}_v, n + \mathcal{W}_v]$ do
3         $J(\Phi[n]) = -\log \Pr(\Phi[n_i]|\Phi[n])$
4         $\Phi[n] = \Phi[n] - \frac{\partial J(\Phi[n])}{\partial \Phi[n]}$
5     end
6 end
```

that these representations are influenced by present and past neighbors, by dividing it into two sub-problems. We learn the spatial representation from the current graph, G_t, and the temporal representation from past graphs $G_{t-1}, \cdots, G_{t-\tau}$, separately. The final trajectory representation is obtained by combining the spatial and temporal representations of a given node.

The STWalk2 algorithm first performs a random walk in the current time step graph G_t, called as *SpaceWalk*, and learns the spatial representation for each node using the SkipGram algorithm. $\Phi^{space}_{u,:}$ corresponds to the learned spatial representation of node u. To learn the temporal representation of a node, STWalk2 constructs G_{new}, which contains only one node from current graph and neighbors from past graphs. For example, in order to construct such a G_{new} in Figure 4.1, we remove all neighbors from current time-step graph of node u_t. The remaining graph will have u_t linked only to past subgraphs of u_{t-1}, u_{t-2} and u_{t-3}. This will be used as G_{new} to learn the temporal representation (in a manner similar to learning the spatial representation above).

The random walks on G_{new} capture the temporal relationship between the current node and nodes from its past graphs. $\Phi^{time}_{u,:}$ corresponds to the learned temporal representation of node u. Finally, the two learned embeddings are combined to get trajectory representation of nodes. Algorithm 6 summarizes this approach. Line 15 of Algorithm 6 states that the final trajectory representation is a function of spatial and temporal representations. In this work, we found vector addition to work very well in practice in the experiments.

Thus, in STWalk2, we ensure that the trajectory representation of the current node u_t has influence from past nodes by learning the temporal representation explicitly. Learning the temporal representation separately forces the random walk to explore the past graphs of the same node and capture the temporal information between the current node and its past. Therefore, STWalk2 is

Algorithm 5: STWalk1

```
Input: Given node $n \in [1, N]$,
time step $t \in [1, T]$,
length of random walk: $\mathcal{L}$,
temporal window size: $\tau$,
vocabulary window size: $\mathcal{W}_v$,
Set of graphs $\mathcal{G}$:$\{G_t, G_{t-1}, \cdots, G_{t-\tau}\}$, size of embedding: $d$,
number of restarts (starts at same node): $\rho$
Output: Updated $n^{th}$ row of matrix of $\Phi_t$ of size: $N \times d$
1 Initialize $\Phi_t[n] \in \mathbb{R}^d$
2 for $i=0$ to $\rho$ do
3     new_graph = createSpaceTimeGraph($\mathcal{G}$,$\tau$,$n$)
4     spaceTimeWalk = randomWalk(new_graph,n, $\mathcal{L}$);
5     $\Phi_t[n]$ = SkipGram ($\Phi_t$, spaceTimeWalk, $\mathcal{W}_v$);
6 end
```

able to learn richer trajectory embeddings than STWalk1 because in STWalk1 we consider the present and past neighbors together, giving the random walks an option of skipping few nodes from past graphs. Experimental results in Table 4.2 validate that the trajectory embeddings learned by STWalk2 perform better than embeddings learned by STWalk1. We now present details of our experiments.

4.3 Experiments

We discuss details of our datasets in Section 4.3.1. We evaluated the proposed methods primarily on the task of trajectory classification. We also extended this to study the goodness of the learned trajectory representations through visualization as well as a second task, change point detection. Their performance is compared with three baseline methods, which are described in Section 4.3.2. Section 4.3.3 and 4.3.4 provide the details of the experimental setup and results for the trajectory classification and change point detection tasks respectively. The code and datasets used for experiments are available on github[1].

4.3.1 Datasets

We use three real-world datasets with temporal graphs for the experiments in this work. Each of these datasets is described below.

DBLP Dataset: The DBLP[2] dataset contains bibliographic information about a large collection of computer science publications. This is a dataset commonly used in graph analysis. We have considered a subset of publications which fall under four areas of research: Database, Artificial Intelligence, Data Mining and Information Retrieval. We construct graphs from the this subset of DBLP dataset by considering authors as nodes and connecting two nodes if the corresponding authors have published a paper together. We constructed 45 such annual graphs starting from year 1969 till 2016, excluding years 1970, 1972 and 1974 (whose data was not available on the DBLP

[1]https://github.com/supriya-pandhre/STWalk
[2]https://www.informatik.uni-trier.de/ ley/db/

Algorithm 6: STWalk2

```
Input: Given node $n \in [1, N]$,
time step $t \in [1, T]$ Set of graphs $\mathcal{G}$:$\{G_t, G_{t-1}, \cdots, G_{t-\tau}\}$,
length of spatial random walk : $\mathcal{L}_\alpha$,
length of temporal random walk : $\mathcal{L}_\beta$,
temporal window size: $\tau$,
vocabulary window size: $\mathcal{W}_v$,
size of embedding: $d$,
number of restarts (starts from same node): $\rho$
Output: Updated $n^{th}$ row of matrix of $\Phi_t$ of size: $N \times d$
1  Initialize $\Phi_t[n] \in \mathbb{R}^d$
2  Initialize $\Phi^{space}$, trajectory representation matrix of nodes only at $G_t$
3  Initialize $\Phi^{time}$, trajectory representation matrix of past nodes at $G_{t-1}, \cdots, G_{t-\tau}$
4  for i=0 to $\rho$ do
5      //for space walk
6      spaceWalk = randomWalk($G_t$, $n$,$\mathcal{L}_\alpha$)
7      $\Phi^{space}[n]$ = SkipGram($\Phi^{space}$, $spaceWalk$,$\mathcal{W}_v$)
8      //for time walk
9      Construct $G_{new}$ as described in Sec 4.2.2
10     timeWalk = randomWalk($G_{new}$, $n$,$\mathcal{L}_\beta$)
11     $\Phi^{time}[n]$ = SkipGram($\Phi^{time}$, $timeWalk$,$\mathcal{W}_v$)
12     //final trajectory embedding
13     $\Phi_t[n] = f(\Phi^{space}[n], \Phi^{time}[n])$
14 end
```

website).

Epinion Dataset: Epinion[3] is a popular product review site where users write critical reviews about products from various categories. The Epinion dataset[4] contains review details such as user ids, product ids, category ids, time-stamp when the ratings were created, along with few other fields. This dataset has been used in earlier work such as [58] and [59]. We create a dynamic network from this dataset by considering users as nodes and inducing an edge between two users if they have rated products from the same category. We created 110 such graphs from monthly data starting from March 2002 until April 2011.

Ciao Dataset: Ciao[5] is another popular product review site. This dataset[6] contains fields similar to those in Epinion: user ids, product ids, category ids, and time-stamps when the ratings were created. The temporal graph for the Ciao dataset also contains user-ids as nodes and edges between nodes when the corresponding users review products from the same category. We created 115 such graphs from monthly data starting from September 2001 until March 2011.

More details of each of these datasets are shown in Table 4.1.

[3]Website: http://www.epinions.com.
[4]http://www.cse.msu.edu/ tangjili/trust.html
[5]http://www.ciao.co.uk
[6]http://www.cse.msu.edu/ tangjili/trust.html

Datasets	No. of nodes	No. of edges	Duration	No. of Trajectories
DBLP	118,030	288,634	1969-2016	15,400
EPINION	21,575	2,590,798	Mar 2002-Apr 2011	21,575
CIAO	2249	27,209	Sep 2001-Mar 2011	1800

Table 4.1: Statistics of datasets: number of nodes, number of edges, the duration of data considered for analysis and number of node trajectories whose class labels are known. This is further described in Sections 4.3.1 and 4.3.3.

4.3.2 Baseline Methods

We compare the performance of our proposed methods against the following baseline algorithms.

Node PageRank: This method defines a simple approach to use graph structure to study changes in behavior of nodes over time, viz. node trajectories. In this method, we compute the PageRank value of nodes at each time step of a graph within a window. The vector containing these PageRank values at different time steps is used as a trajectory representation for the tasks evaluated in this work.

Avg DeepWalk: DeepWalk [27] uses a truncated random walk to represent each node in a static graph. To compare performance of STWalk against DeepWalk, we run the DeepWalk algorithm at each time step. The average of these node embeddings is considered as trajectory representation of the node across time.

LSTM-AE: Inspired by the work in [60] to learn representations of video sequences using an unsupervised approach, we use a similar approach for learning node trajectories in a temporal graph. DeepWalk [27] is used to learn d-dimensional node representations at each time-step. A Long Short-term Memory (LSTM) AutoEncoder (AE) is used to learn trajectory representations (similar to [60], which used this approach for videos though). The encoder-LSTM takes the sequence of d-dimensional node embeddings and encodes the sequence, and the decoder-LSTM reconstructs the original sequence from the embedding obtained using the encoder. Once this LSTM-AE network is completely trained, the output of the encoder-LSTM is used as the trajectory representation. This baseline algorithm was implemented using Keras [61], and trained using the Adam optimizer with the mean square error loss function. The low-dimensional embedding obtained through encoder was chosen to be of $d/2$ dimensions after empirical studies.

4.3.3 STWalk for Trajectory Classification

We studied the effectiveness of the representations of trajectories learned using the proposed methods on the task of trajectory classification (where the node's trajectory in a pre-specified time window is considered). We now describe how each of the aforementioned datasets were studied in this work for trajectory classification, along with the corresponding results. We note that in case of each of the datasets, the total number of node trajectories are divided into a 70% training set and the remaining 30% as a test set. An ensemble voting-based classifier (comprised of random forests and Support Vector Machines) was trained on the training set using stratified k-fold cross-validation with $k = 10$ to finetune the hyperparameters of the classifier (this classifier was chosen after empirical studies,

		Methods				
		Node PageRank	Avg DeepWalk	LSTM-AE	STWalk1	STWalk2
Emb Dim	Win Size	Acc (%)	Acc (%)	Acc (%)	Acc (%)	Acc (%)
		DBLP				
64	5	70.42	72.61	73.89	77.66	**78.74**
64	10	61.94	70.21	73.31	70.43	**77.53**
128	10	74.53	71.53	74.32	**83.72**	78.25
		EPINION				
64	5	50.03	51.44	50.08	51.82	**53.10**
64	10	53.24	52.21	50.50	55.51	**56.70**
128	10	57.36	53.79	52.70	59.03	**62.03**
		CIAO				
64	5	51.94	52.02	54.17	56.31	**57.27**
64	10	25.56	24.65	43.95	58.17	**62.20**
128	10	32.03	53.15	53.80	57.15	**60.29**

Table 4.2: Trajectory classification results on the DBLP, EPINION and CIAO datasets. STWalk1 and STWalk2 are the proposed algorithms, while Node PageRank, Avg DeepWalk and LSTM-AE are the baseline methods. Column 'Emb Dim' indicates the dimension of the trajectory representation, while 'Win Size' indicates the size of time window considered for learning trajectory representations.

and is also known popularly to be a very robust classifier). The classifier with the best hyperparameters is then trained on the complete training dataset to obtain the final model, and used to study performance on test data. We use the classification accuracy as the performance metric for this task. We repeat the process five times and report the average performance in our results (Table 4.2). This process is maintained for each of the considered datasets.

DBLP Dataset: To evaluate our model on the temporal graphs in the DBLP dataset, we considered non-overlapping windows of 5 years, leading to 9 such windows over the 45 graphs in this dataset. (For example, while learning the trajectory embedding of nodes at time t_5, we consider the graph at t_5 as well as the previous 4 graphs: t_4, t_3, t_2 and t_1.) A similar process is used for window of 10 years too.

The representations of author trajectories learned using the proposed methods and baseline methods are then used for the binary classification task, where class labels are "Expert Researcher" and "Interdisciplinary Researcher". These labels are assigned to each node trajectory in the following manner. Based on the conference where each author has published his/her work, we assign one of the labels from the following to each node of a graph at each time step. Label *Database*: if published in ICDE, VLDB, SIGMOD, PODS or EDBT; label *Artificial Intelligence*: if published in IJCAI, AAAI, ICML, or ECML; label *Data Mining*: if published in KDD, PAKDD, ICDM, PKDD or SDM; label *Information Retrieval*: if published in SIGIR, WWW, ECIR or WSDM. Subsequently, the trajectory labels are determined based on the node labels assigned at each time-step. If the node label is constant throughout the considered time window, the trajectory label corresponding to that node is "Expert Researcher". If the node label changes within the time window, we assign the label "Interdisciplinary Researcher" to the author trajectory.

STWalk has four hyper-parameters: *embedding dimension:* dimension of the trajectory representation; *window size:* number of time steps to consider while learning trajectory embedding; *walk length:* length of random walk performed on the spatio-temporal graph; and *restarts:* number of random walks starting from the same node. Table 4.2 shows the performance of the proposed and baseline methods in three different hyper-parameter settings. The *walk length* value was set to 30, and the *restarts* value to 40.

Epinion Dataset: In this dataset, when we set the hyperparameter, window size, to five, 22 windows are considered for analysis (among the 110 graphs available in this dataset), and similarly when the window size is set to 10. Category IDs are used as node labels, which are subsequently used to determine the ground truth labels for node trajectories. If a user has reviewed products from the same category during all the time steps considered within a time window, the trajectory corresponding to the user will be assigned the label "Proficient reviewer", indicating that the user has good knowledge about the products in the category. If the user reviews products from different categories during the considered time window, the user is assigned the label "Versatile reviewer", indicating that the user has expertise in assessing products from many categories. Thus, trajectory classification is modeled as a binary classification problem on this dataset, and we report the result of executing our algorithm on Epinion dataset in various hyper-parameter settings in Table 4.2. Similar to the DBLP dataset, we set the hyper-parameters *walk length* to 30 and *restarts* to 40 for this dataset.

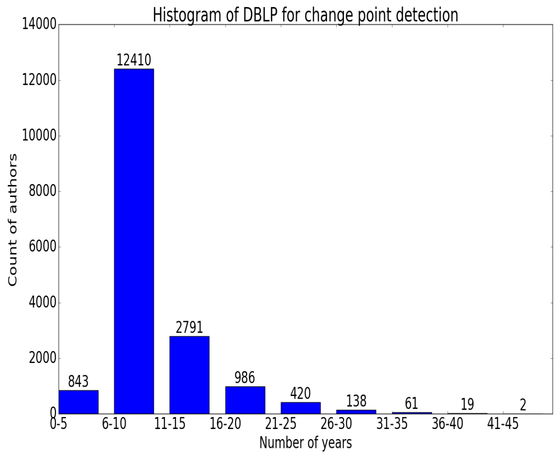
Ciao Dataset: In this dataset, when we set the hyperparameter, window size, to five, 23 windows are considered for analysis (among the 115 graphs available in this dataset), and similarly when the window size is set to 10. The category ID of the product that a user reviews is considered as the node label, and ground truth trajectory labels are determined based on these node labels. Similar to the Epinion dataset, if the node labels remain constant for the entire time window, the corresponding node trajectory is assigned the label "Experienced reviewer", and if the user reviews products from multiple categories within the same time window, the node trajectory is assigned the label "Multifaceted reviewer". Binary classification is then used to validate the effectiveness of trajectory representations learned on this dataset. The *walk length* and *restarts* hyperparameters are chosen as on the Epinion dataset.

Table 4.2 shows the classification accuracies of the proposed and baseline methods on all the three datasets. The proposed algorithms, especially STWalk2 shows superior performance in most settings. In case of the CIAO dataset, the proposed algorithms show significant increase in accuracy over the baseline methods. Interestingly, the LSTM-AE method, which was introduced as a baseline method in this work, also performs better than the other two baseline methods. The results support the proposed methods in general. In particular, the improved performance of STWalk2 shows that having separate space and time walks provides better representations. We surmise the reason for this to be the improved coverage of temporal neighborhoods through this approach when compared to STWalk1 under the same random walk lengths.

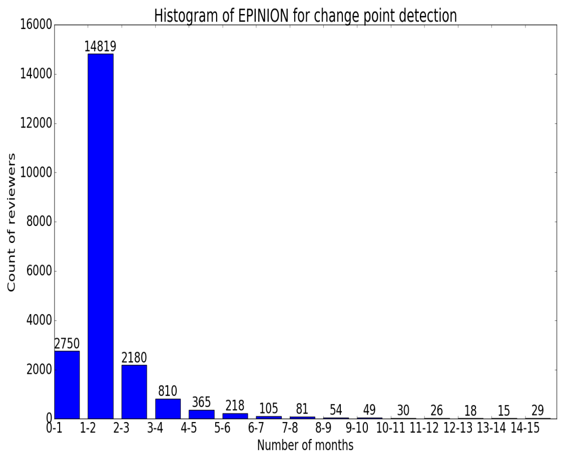
4.3.4 STWalk for Change Point Detection

In a temporal setting, analyzing change points can be critical in applications such as concept drift and anomaly detection. As the trajectory representation captures the change in behavior of nodes in temporal graph over a given time period, this makes the trajectory representations suitable features to study such change points.

In the DBLP dataset, we use such an approach to find the average time spent by an author in his/her research area before switching to another domain, thus finding the 'points of change in research domain' in the author's trajectory. Specifically, we consider the trajectory labels across multiple windows of the author (as obtained from Section 4.3.3). We study the sequence of trajectory labels and based on the pattern of occurrence of "Interdisciplinary" and "Expert" labels, we calculate the average duration of time spent in a single domain. Figure 4.2a shows the histogram of authors who spent certain number of years in one domain before working in other research area. This analysis allows us to answer queries such as the minimum (or average) number of years spent before a researcher becomes interdisciplinary. For example, Figure 4.2a shows evidently that the average duration of time before a researcher becomes interdisciplinary is between 6-10 years, which agrees with common understanding. We also used the histogram to analyze cases of individual authors, and found the change points to be fairly accurate. This supports the effectiveness of the trajectory representations learned by the proposed methods (STWalk2 was used in these studies, considering its superior performance in the earlier experiments).



(a) Histogram shows the count of authors vs number of years spent by any author in one research area before switching to other domain



(b) Histogram of number of users vs average duration spent on reviewing product of same category before switching to other category.

Figure 4.2: Results of change point detection on the DBLP and Epinion datasets

Similarly, analyzing change points in the Epinion dataset gives insight into the behavior of reviewers. Figure 4.2b shows the trends for the Epinion dataset. It shows the histogram of reviewers and the number of months they review products from one single category, before switching to products from other categories. In this case, it is clear from the figure that a reviewer, on an average, writes reviews for products from one category for 1 to 2 months before switching to other categories, which matches with common understanding too. This corroborates the effectiveness of the proposed method (STWalk2 in this case again) for learning trajectory representations in temporal graphs.

4.3.5 Arithmetic Operations on Trajectory Representations

Mikolov et al. [62] demonstrated that arithmetic operations on learned word representations show a linear structure in embedding space, in the case of text processing. For example, vector("Paris") - vector("France") + vector("England") gives a vector representation nearest to the word embedding for London. Here, we examine whether similar behavior is shown by the learned trajectory representations. We perform this study with the representations learned on the DBLP dataset. In particular, we consider the representations for interdisciplinary authors (shown as yellow points in Figure 4.3, best viewed in color) and expert authors (shown as cyan points in the figure). We then perform the following difference operation: trajectory("interdisciplinary author")-trajectory("Expert author") and examine the resulting representations (plotted in red color in the figure).

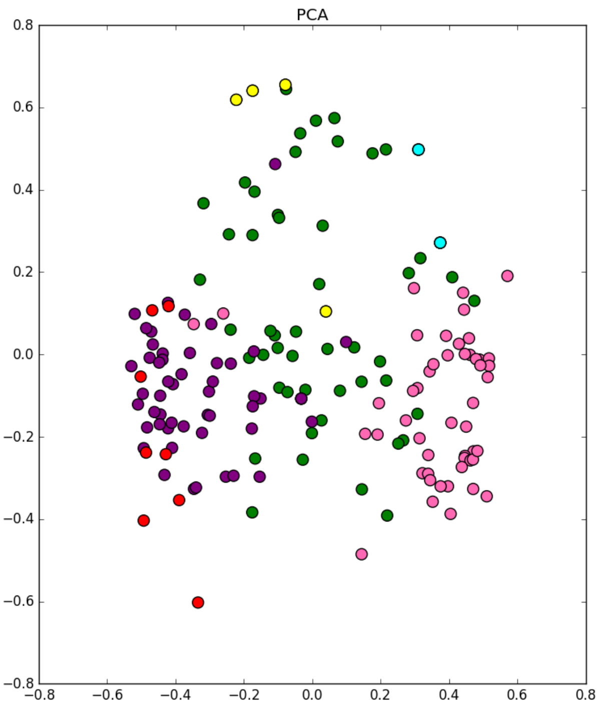


Figure 4.3: Analyzing the goodness of trajectory embeddings of DBLP dataset(Best viewed in color). (i) Arithmetic operations are valid on trajectory embeddings. E.g. (interdisciplinary embedding)-(specialized embedding)=(other specialized embedding), i.e., taking arithmetic difference as in (AI+DB) (yellow points)-DB (cyan points)=AI (red points). (ii) Pink color cluster represents DBLP authors working in Database (DB). Violet color represents authors working in Artificial Intelligence (AI) and the green color cluster indicates the interdisciplinary authors working in both areas AI and DB.

Figure 4.3 shows the Principal Component Analysis (PCA) plot for the result of performing arithmetic difference operation on the trajectory representations of DBLP authors, as described above. For this plot, we considered only the trajectory vectors of authors working in the *Database* (DB) domain (shown in pink color on the scatter plot). We also considered the trajectory embeddings

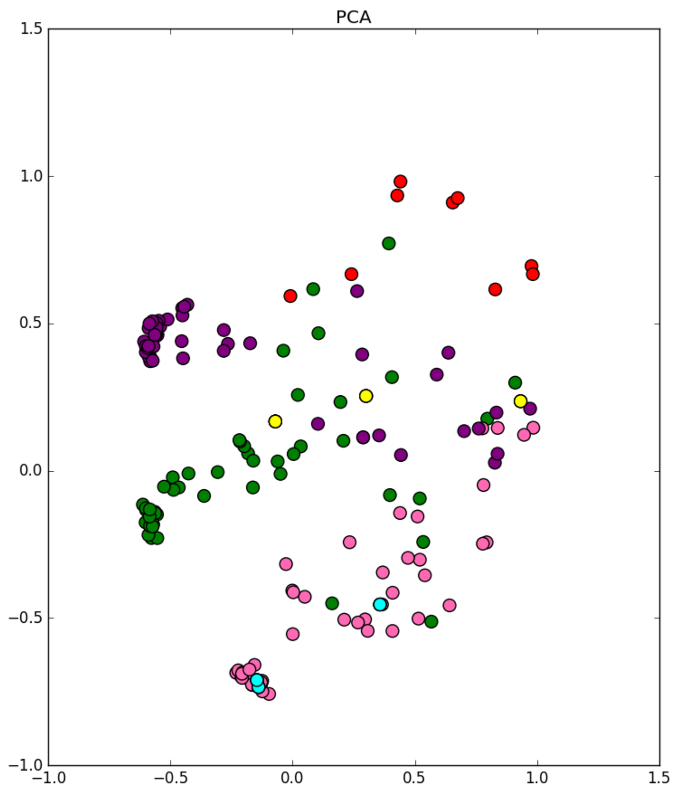


Figure 4.4: Analyzing the goodness of trajectory embeddings of Epinion dataset (Best viewed in color). (i) Arithmetic difference of users' trajectory who review products from category (Movies+Books) (yellow points)-Books (cyan points)= Movies category (red points). (ii) Pink color cluster represents users who review Books. Violet color represents users who review Movies and the green color cluster indicates the versatile users who review products from both categories Movies and Books.

of authors who focus only on the *Artificial Intelligence* (AI) domain (violet color on the scatter plot). Thus, the pink and violet clusters represent authors with trajectory label "Expert Researcher". To evaluate the goodness of learned trajectory embeddings, we consider the representations for authors with labels "Interdisciplinary Researcher". In particular, we consider trajectories of authors who have published in both *Database* and *Artificial Intelligence* (AI+DB) venues. These interdisciplinary authors are shown in green color data points on the scatter plot. Three distinct clusters, in Figure 4.3 indicating DB (pink), AI (violet) and AI+DB (green), validate that the proposed model STWalk is able to learn rich trajectory embeddings for "Expert" authors and "Interdisciplinary" authors.

To validate if the arithmetic operation: (AI+DB)-DB=AI holds for the learned trajectory embeddings, we perform the following steps. We take the trajectory vectors of a randomly picked "Interdisciplinary" author from the collection of vectors mentioned above. The selected "Interdisciplinary" points are shown in yellow color in Figure 4.3. These are part of the green cluster, which represents all "Interdisciplinary" authors. Similarly, we pick trajectory embeddings of "Expert" authors from a DB background (shown in cyan color points). We then apply vector difference operation on vectors from the above two selected sets, and the resulting vectors are plotted in red color. As seen in Figure 4.3, the resulting red points are near the violet points, which indicates the cluster for "Expert" authors in AI. This demonstrates that simple vector arithmetic on the learned trajectory representations show linear structure and hence, can be used for developing interesting applications.

Figure 4.4 shows the results of similar vector operation on Epinion dataset. We considered trajectory representation of users who review products from category *Movies* and *Books* as "Proficient reviewer", represented by violet and pink clusters respectively in figure 4.4. The trajectory representation of users reviewing products from both categories *Movies* and *Books* as "Versatile reviewer", represented by green cluster. We randomly select trajectory embedding of "Versatile reviewer", the yellow points and "Proficient reviewer", the cyan points, and then perform vector subtraction. The resulting points are plotted in red color. We can conclude from 4.4 that these points are near violet cluster which represents *Movies* cluster, validating that the learned trajectory representations show linear structure

4.4 Conclusion

In this chapter, we proposed STWalk a novel approach to learn trajectory representations in temporal graphs, by capturing change in nodes' behavior over time. We presented two algorithms to learn such trajectory representations: STWalk1 learns the representations by considering the present neighbors and past neighbors together, and STWalk2 models it as two subproblems, solving each separately before combining the results to get the final trajectory embedding. We validated the effectiveness of the learned trajectory representations on three real-world datasets: DBLP, Epinion and Ciao. We compare the results against three baseline algorithms, and show that the proposed STWalk outperforms baseline methods in various settings.

Furthermore, we demonstrated that simple arithmetic operations are valid on the trajectory representations learned through the proposed methods, i.e., the vector difference: trajectory("interdisciplinary author")- trajectory("specialized author")=trajectory("other specialized author"). We also examined the usefulness of trajectory embeddings in change point detection. Each of these experimental studies showed the effectiveness of trajectory representations learned by STWalk.

Chapter 5

Attend, Order and Classify: End-to-End Deep Neural Network for Graph Classification

5.1 Motivation

With the emergence of deep learning methods such as Convolutional Neural Networks and Recurrent Neural Networks, many breakthroughs have been achieved in the computer vision, natural language processing and speech processing domains. An important characteristic of these deep learning techniques is their ability to learn the important features that are necessary to excel at a given task on their own, unlike traditional machine learning algorithms which are dependent on handcrafted features. Considering these facts, researchers have started working on extending deep learning concepts, particularly Convolutional Neural Network for graphs. An early effort of generalizing neural networks for graphs is the work by Gori et al [63] and Scarselli et al [64]. Shuman et al [65] used the Convolution Theorem (convolution in spatial domain is equivalent to multiplication in Fourier domain) to learn the spectral filters by computing the Laplacian of a given graph. Defferrard et al [66] proposed an algorithm that learns approximate spectral filters by recursively computing the Chebyshev polynomial. Kipf et al [67] further simplified this approach by restricting it and considering only 1-hop neighbors. The major drawback of spectral methods is the dependency of spectral filters on eigenbasis of the Laplacian. This means that the filters learned on one graph topology are not valid on graphs with a different topology; in other words, the filters are not generalizable. One solution to this problem is to apply convolution in the spatial domain and our proposed framework *AtOrC* falls in this category by working directly on the graph.

Applying a convolution neural network directly on graph poses two major problems: (i) firstly, the graphs do not have an ordered structure, and hence the nodes in it do not have an unique ordering; this makes it not viable to directly apply convolutional filters. (ii) Secondly, the nodes have variable number of neighbors and this presents the problem of appropriately selecting few nodes and ignoring the rest while performing a single convolution. In other words, before performing convolution, the algorithm needs to select a subset of nodes and order them according to some criteria. A naive

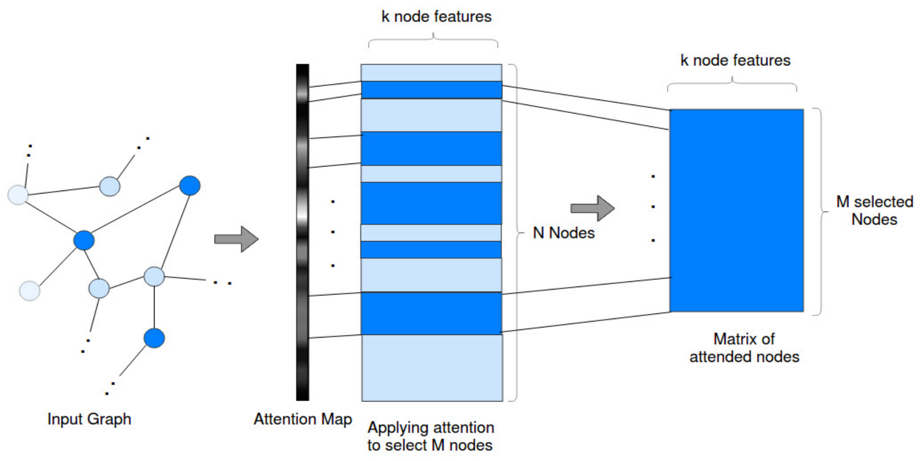


Figure 5.1: Based on which rows of input matrix are picked by the attention map while classifying a graph, we can interpret the decisions made by the neural network by traversing back to nodes corresponding to those rows.

way of achieving this is to use any one of the centrality measure such as node degree and select the nodes with highest node degree and order them in decreasing order of degree. However, this process creates the extra overhead of preprocessing every input graph. Besides, this step is not differentiable and hence cannot be used if we wish to train end-to-end using a convolutional neural network. In this work, we hence propose Attend, Order and Classify, an end-to-end deep learning architecture for graph classification.

The key contributions of our work are as follows:

- There are three modules in our proposed framework: the Soft-Attention module, the Soft-Permutation module and the Convolutional Neural Network. As the name Attend, Order and Classifysuggests, each one performs one task: Attend, Order and Classify respectively. Each of these modules are differentiable, hence allowing backpropagation to be used to train the network. To the best of our knowledge, our proposed framework is the first end-to-end architecture that learns which nodes to focus on and how to order those nodes, in order to help the convolutional layers learn better features and make more accurate decisions on the graph classification task.

- **Soft-Attention Module:** Our framework uses Soft-Attention over the nodes of the graph to select few important nodes. It is achieved by applying 1D Gaussian filter, whose mean and variance are learned by backpropagation algorithm. We also show that these learned attention maps can be used to interpret the decisions made by the model(figure 5.1).

- **Soft-Permutation Module:** We also propose the Soft-Permutation network that learns the best ordering of nodes that will help the model to perform better at graph classification task.

- We use the Convolutional Neural Network to learn the local features of the nodes. Instead of selecting only one filter size for the convolution operation, we used Inception Module, inspired by the InceptionNet [2] architecture.

- The experimental results on several benchmark graph datasets substantiate the effectiveness of the proposed architecture.

The rest of the chapter is structured as follows: In section 5.2 we review work that is four closely related to our algorithm. Section 5.3 explains the algorithm, followed by details of experiments are given in section 5.4. In section 5.5, we report the ablation study and we also discuss here about the use of attention maps in understanding the decisions.

5.2 Related Work

5.2.1 Learning Permutation

Current deep learning algorithms assume that the given input data has a certain order. The CNN-based image classification algorithm cannot predict the correct image class if the input image is shuffled one. The RNN used for machine translation assumes that the input sequence is in the correct order. Hence, we need to make sure that the all input graph have been arranged in the same order. Solving this ordering problem is equivalent to learning a permutation matrix. However, learning a discrete permutation matrix is not feasible and hence, a doubly stochastic matrix: a nearest approximation of permutation matrix, is learned using Sinkhorn-Knopp(SK) algorithm. One of the early efforts in approximating permutation matrix is by Gold et al [68]. The authors pose it as a combinatorial optimization problem to solve weighted graph matching, traveling salesman problem, and graph partitioning problem by using deterministic annealing and Sinkhorn-Knopp iterative algorithm. Recently, Cruz et al [69] proposed a method that learns a correct order of shuffled input image segments by using Sinkhorn Normalization layer. It is a supervised algorithm, where the correct order of the image segments are already known. In case of images, the pixels have predetermined location and order, when we arrange these pixels according to it the whole image gains meaning, and hence the problem of learning permutation matrix can be posed as a supervised problem. However, in case of graphs, the nodes do not have an unique order. Hence, the nodes can be ordered by using any one of the existing graph centrality measures such as PageRank. Instead of ordering nodes according to one specific centrality measure, we propose SoftOrdering module that learns the ordering of nodes using backpropagation such that the learned ordering is the most useful in graph classification.

5.2.2 Interpretability of Neural Network

In the domain of computer vision, different methods have been proposed for understanding decisions of neural network. For example, by looking at the filters of convolution neural network, one can understand which part of image is helping the neural network to come to a conclusion. Other methods include occlusion and gradient based methods. Some of the work in this area uses attention mechanism to see which part of the input data is being used to compute the output. The attention mechanism has been used for image data, text data as well as speech data and they proved to work better at many tasks such as language translation [70], [71], [72], [73], image captioning [74], [75], and visual question answering [76], [77], [78], [79], [80]. A brief survey of work on VQA is given in [81]. Another interesting line of work is using attention mechanism for generating images [82], [83] and videos [84], [85]. Our work is inspired by the DRAW [82]

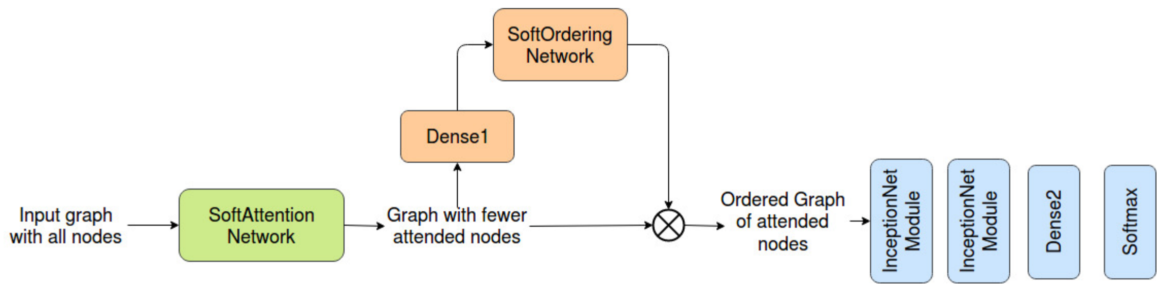


Figure 5.2: AtOrC : complete architecture

5.3 Methodology

Graphs are ubiquitous and used in many applications. Hence, analyzing graphs play a very important role. Particularly in health-care domain, the molecular properties of medicine can be studied by representing them as graphs, where nodes are the represented as atoms and edge between two nodes indicates that the two atoms are linked. Study of these graphs is helpful in learning important properties which will be used to determine whether a molecule is carcinogenic or not. It has been proved that convolutional neural network are good at capturing the local features, combining them to learn higher level representations and use them to perform better at classification. This motivated us to use the convolutional neural network for graph classification. However, the traditional convolutional neural network used in computer vision area is defined for data in Euclidean domain i.e. for the data that can be represented on a regular grid. The graph data falls under the category of non-Euclidean domain and the convolution and pooling operations are not well-defined in non-Euclidean space. Hence, using the convolution operation directly on graphs poses following two challenges:

- *Variable number of nodes:* The convolutional neural network work on the fixed size input images i.e. fixed number of pixels to process. However, in case of graphs, the number of nodes in input graph need not be same. Each example in graph dataset can have different number of nodes. This creates problem, because to apply the convolution operation, we need to fix the number of nodes to be processed before-hand and accordingly either select some of the existing nodes from bigger graph-leading to loss of information or add dummy nodes in smaller graph-introducing unnecessary noise in the input data.

- *No spatial order:* The sequence in which nodes are processed is not unique. This is not the case with images, as each pixel has specific fixed position and when they are in order the convolutional filter is able to learn meaning patterns. However, the nodes of the graphs can be arranged in $n!$ ways and hence the adjacency matrix that is used for graph classification. This creates problem to convolution operation in learning features, as it is good at capturing the local features when the input is in order.

To tackle above mentioned problems we propose the SoftAttention and the SoftOrdering modules. The complete network architecture of AtOrC is given in figure 5.2.

5.3.1 Problem Statement

Given a graph $G : (V, E)$ with set of vertices $V = \{v_1, v_2, \cdots, v_N\}$ of size N and E is a set of edges. Each node has feature vector of size k and the graph is represented as a matrix of node features X of size N x k. The task of graph classification is to learn a function $f : G \to y$, where $y \in Y$ indicates a label of given graph G from a set of all labels Y.

The AtOrC takes an entire graph as input and gives it to the SoftAttention module that selects important M nodes from the set of all nodes in the graph, such that $M < N$. The output matrix of SoftAttention module, denoted by X', represents attended graph of size M x k. This matrix is given as input to SoftOrdering module, which learns the permutation matrix P of size M x M that indicates correct order of these nodes necessary for the convolution neural network. After performing matrix multiplication of the permutation matrix P and graph X', we get *attended* and *ordered* graph X'' of same size as X'. This graph is given to the convolution neural network which learns the features useful for graph classification. The details of the SoftAttention and SoftOrdering module are given following subsections.

5.3.2 SoftAttention Module

To select M nodes from N nodes which are most helpful in classifying the graph, we learn the attention map on the nodes. This attention map consists of M number of 1D Gaussian filter. Due to the nature of Gaussian distribution, each filter has the soft glimpse over the nodes of the graph and hence the name of the module *SoftAttention*. One could use Bernoulli distribution to achieve the hard attention. However, to make the module differentiable and in turn backpropable, we chose to use the Gaussian distribution.

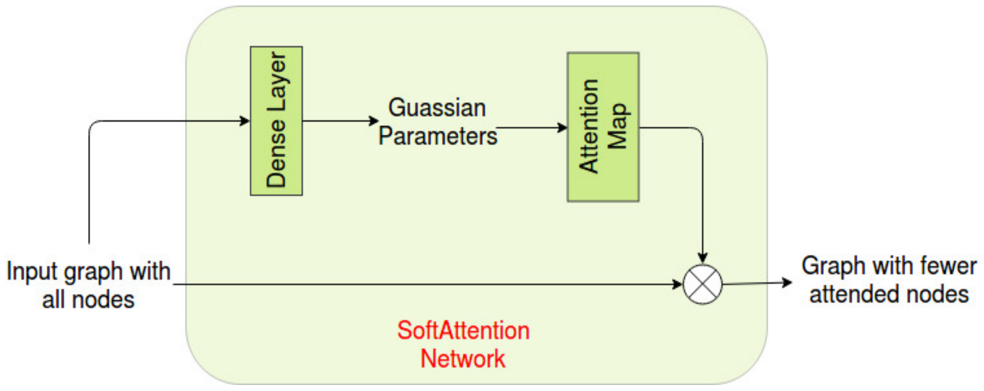


Figure 5.3: Architecture details of SoftAttention module

The selection of M nodes from N can be restated as the neural network is having a attention glimpse size of M. The parameters of these filters i.e. means $\tilde{\mu}$ and variances $\tilde{\sigma^2}$ are learned by the neural network, as given in equation 5.1. To make sure the learned means and variances are positive, we take the exponential. In equation 5.2, we first rescale the means to make sure they are in range $[0, 1]$ by applying sigmoid function, denoted by $sigm$. it is then multiplied with the N i.e. the number of nodes in original input graph to see which node out of N this filter has selected.

$$\tilde{\mu}, \tilde{\sigma^2} = W_{attn}X + b_{attn} \tag{5.1}$$

$$\mu = N * sigm(exp(\tilde{\mu})) \tag{5.2}$$

$$\sigma^2 = exp(\tilde{\sigma^2}) \tag{5.3}$$

The matrix A, in equation 5.4, represents the learned attention map of Gaussian filters. The index i indicates filter number which takes values in the range $[0, M-1]$ and index n indicates the node location that takes values in range $[0, N-1]$. Hence, the equation 5.4 calculates importance of node n with respect to the Gaussian distribution centered at i. This map, of size M x N, is applied on the input graph X, of size Nxk, to get the reduced output graph X', of size M x k, as given by the equation 5.5. X' indicates the graph with only important nodes. Once we get the attended graph X', it is given to SoftOrdering module, which learns to order the attended nodes.

$$A[i, n] = exp\left(\frac{-(n-\mu_i)^2}{2\sigma_i^2}\right) \tag{5.4}$$

$$X' = A * X \tag{5.5}$$

5.3.3 SoftOrdering Module

Once we know which nodes to focus on, it is important to know in which order we need to arrange them, so that it is easy for convolution neural network to learn local features. The main aim of the SoftOrdering module is to generate a permutation matrix that tells about the ordering of nodes, when ranked accordingly, it will help the filters in convolution neural network to learn necessary features. Permutation matrix P is a special square matrix where each element can be $0, 1$ and the ones are placed in such a way that, each row and column will have only one 1. As it is an Integer Programming problem, learning exact permutation matrix needs discrete optimization methods which are non-differentiable. The closest approximation of permutation matrix is doubly stochastic matrix. To learn an approximation of permutation matrix by using the neural network, we have to pretrain it on matrix with values sampled from normal distribution and plug the learned weights into main network to order the nodes. The details of the process are as follows:

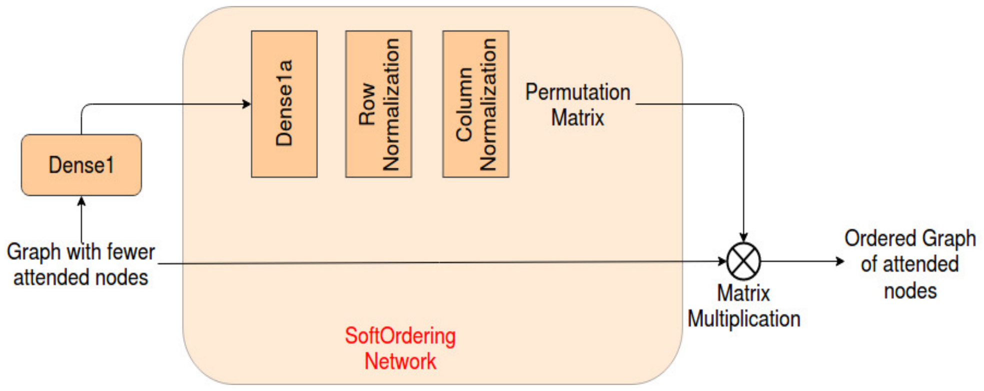


Figure 5.4: Architecture details of SoftOrdering module

Pretraining the SoftOrdering network:

Inspired by the SoftAssign [68] algorithm, that solves the assignment problem by finding an approximate permutation matrix. The algorithm uses deterministic annealing followed by Sinkhorn-Knopp algorithm. The Sinkhorn-Knopp algorithm [86], [87] is an iterative algorithm that converts a given positive square matrix into a doubly stochastic matrix which is close to final assignment matrix (i.e. permutation matrix). In our case, however, we need a function that mimics this iterative process in a single step. As neural networks are proven to be good at learning function approximations, we used a simple neural network with an input layer and a output layer for this task. The input to the network is a matrix for which we want to learn an approximate permutation matrix and output of the network is compared with the output obtained by iterative process. We used square loss function to train the network. The trained weights of this network are then used to initialize the weights of *Dense1a* layer in SoftOrdering module. The network architecture used is given in figure 5.4.

To make the neural network used for pretraining process data agnostic, we generated the input data matrix whose elements are sampled from normal distribution and we used the iterative process of converting a matrix to doubly stochastic matrix to generate the output matrix, as given by the algorithm 7. Input to the algorithm is the normally distributed matrix Z. We scale the matrix multiplying by β and then take the exponentials, as suggested by Gold et al [68]. The β is hyperparameter and bigger the value, closer the output matrix to the true permutation matrix. Steps 3 to 12 in algorithm 7 shows the Sinkhorn-Knopp iterative algorithm, which performs alternate row and column normalization to obtain doubly stochastic matrix. The algorithm is guaranteed to converge in a finite number of iterations [86] [87].

Algorithm 7: Algorithm to generate permutation matrix from given matrix using Sinkhorn-Knopp algorithm [86], [87]

```
   Input  : $Z$:Matrix with values from Normal distribution of size ($M$ x $M$)
   Output: $P^{max\_iter}$:Permutation matrix after $max\_iter$ iterations of size ($M$ x $M$)
1  Init: $\beta = 10.0$, $iter = 1$
2  $P^0 = exp(\beta Z)$
3  while $iter < max\_iter$ do
4      // Row normalization
5      for $i = 0$ to $M$ do
6          $P_{ij}^{iter} = \frac{P_{ij}^{(iter-1)}}{\sum_{j=0}^{M} P^{(iter-1)}{}_{ij}}$
7      end
8      // Column normalization
9      for $j = 0$ to $M$ do
10         $P_{ij}^{iter} = \frac{P_{ij}^{(iter)}}{\sum_{i=0}^{M} P^{(iter)}{}_{ij}}$
11     end
12 end
```

Learning to order the graph nodes:
The pretrained weights of the neural network are plugged into the *Dense1a* layer of the SoftOrdering module. The input to the SoftOrdering module is the input graph matrix of size Mxk. The *Dense1* layer transforms the input graph matrix into MxM square matrix which will be used to determine the ordering of the nodes. In equation 5.6, W_p indicates the transformation matrix.

$$P = W_{ord}(W_p X') + b_{ord} \tag{5.6}$$

$$X'' = P * X' \tag{5.7}$$

The *Dense1* layer makes sure that the approximate permutation matrix, generated by the SoftOrdering module, is dependent on the input graph. So that, based on how nodes are arranged in the input graph, *Dense1* layer, along with SoftOrdering module, learns the correct ordering of nodes. As the learned permutation matrix is an approximation of true permutation matrix and it has real values, rather than integer values $0, 1$, with near 1 value in one of the cell and near zero values in other cells of the matrix, hence, we call it the *SoftOrdering* module. The output of this module is SoftOrdering matrix P of size MxM. To order the nodes according to this, the input matrix X' is multiplied by P. This gives an attended and ordered graph X'' of size Mxk. It is then given to the classifier module, which includes Convolutional Neural Network, Dense layers and Softmax layer.

5.4 Experiments

We validate the effectiveness of the proposed framework on the Graph Classification task.

5.4.1 Datasets

We have performed experiments on 6 standard benchmark datasets, out of which 4 are bioinformatics and 2 are social network graph datasets[1]. The bioinformatics datasets used in the experiment are: PROTEINS, PTC, D&D, and MUTAG. The PROTEINS datasets has 1113 graphs and in each graph nodes represent secondary structure elements. If two nodes are neighbors of amino-acid, they share an edge. It is an imbalanced dataset with two classes. PTC dataset has 344 graphs of chemical compounds that indicate carcinogenicity of male and female rats. D&D dataset contains 1178 graphs which are classified into two categories: enzymes and non-enzymes. MUTAG is collection of 188 nitro-compounds categorized into two classes based on their mutagenic effect.

To test the validity of the proposed framework, we have also experimented on social graph datasets. Particularly, we have used IMDB movie collaboration dataset. IMDB-B is balanced graph dataset of 1000 examples with 2 classes. Whereas, IMDB-M has 1500 examples categorized into 3 classes. The details of the dataset are given in table 5.1. For all the above datasets, we have used four node features($k = 4$): node degree, node pagerank, average neighborhood node degree and average neighborhood pagerank.

Table 5.1: Properties of graph datasets

Datasets	**No. of graphs**	**Classes**	**Avg nodes**	**Max nodes**
PROTEINS	1113	2	39.05	620
PTC	344	2	14.28	64
D&D	1178	2	384.32	5748
MUTAG	188	2	17.93	28
IMDB-B	1000	2	19.77	136
IMDB-M	1500	3	13.00	89

[1]All datasets are downloaded from https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

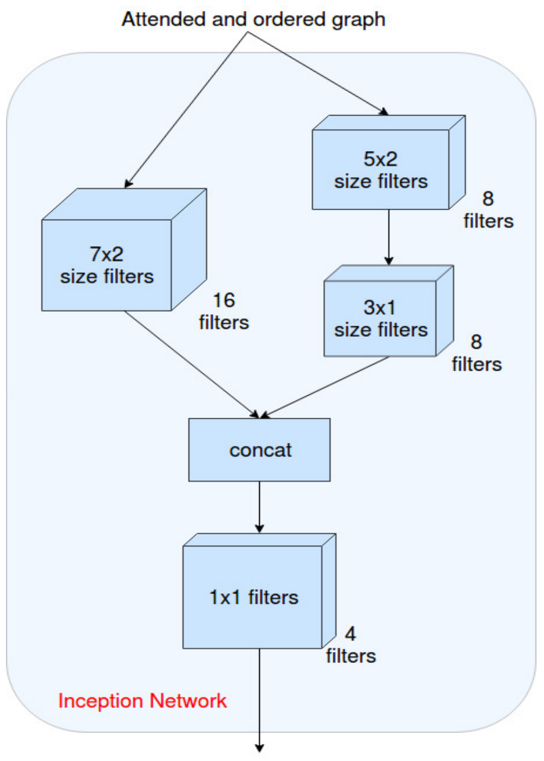


Figure 5.5: Architecture details of one Inception module. Such two inception module have been used in final architecture for graph classification.

5.4.2 Experimental Setup

We compare AtOrC with two types of algorithms: Kernel based approaches and Deep Learning Based approaches. The upper section of table 5.2 shows the comparison of AtOrC with kernel based approaches; Shortest-Path Kernel [39], Random-walk Kernel [40], Weisfeiler-Lehman Kernel [42], and Graph Kernel [41]. The lower section of table 5.2 shows the comparison with Deep Learning based approaches, namely PATCHY-SAN [43], Deep Graph Kernel(DGK) [45], Subgraph2vec [44] and DGCNN [46]. We have used stratified 10-fold cross-validation with 9 folds for training and 1 fold for testing. We have created such ten 10-fold cross-validation data. For each cross-validation data the experiment is repeated 10 times and hence, for each dataset, average of 100 experiments is reported in table 5.2.

The input to the AtOrC is a complete graph. However, different graphs have different sizes and hence the input matrix can be of different size. However, the input to neural network needs to be of fixed size. Hence, to tackle this problem, we have used a fixed size of the matrix that is equal to the maximum number of nodes in the dataset. The reason for fixing the size to max_node is twofold. i) Any value that is less than max_node forces us to remove the nodes from bigger graph and it needs a criteria according to which we can remove the node. It creates preprocessing overhead. ii) The sole purpose of the SoftAttention module is to learn the criteria of selecting the important nodes. To perform that, it needs access to all nodes of the graph. Fixing the input matrix size to max_node, helps the SoftAttention module to perform better. Hence, each graph of a given dataset will be represented in a matrix of size $N = max\_node$ x k, irrespective of its actual size. For graphs

Table 5.2: Classification accuracy comparison with existing deep learning algorithms on benchmark datasets (M:attention glimpse size)

	Datasets	**PROTEINS**	**PTC**	**D & D**	**MUTAG**	**IMDB-B**	**IMDB-M**
Kernel based approaches	SP [39]	75.07±0.54	58.53±2.55	-	85.79±2.51	-	-
	RW [40]	74.22±0.42	57.26±1.30	-	83.68±1.66	-	-
	WL [42]	72.92±0.56	56.97±2.01	77.95±0.70	80.72±3.00	-	-
	GK [41]	71.67±0.55	57.32±1.13	78.45±0.26	81.58±2.11	65.87±0.98	43.89±0.38
Deep Learning based approaches	PATCHY-SAN [43]	75.89±2.76	62.29±5.68	77.12±2.41	**92.63±4.21**	71.00±2.29	45.23±2.84
	DGK [45]	75.68±0.54	60.08±0.46	-	87.44±2.72	66.96±0.56	44.55±0.52
	subgraph2vec [44]	73.38±1.09	60.11±1.21	-	87.17±1.72	-	-
	DGCNN [46]	75.10±2.72	**65.43±3.14**	77.21±0.65	85.83±1.66	70.03±0.86	47.83±0.85
Proposed approach	**AtOrC (INCEP,M=10)**	71.99±8.94	55.71±7.58	76.59±3.46	89.99±8.94	64.79±5.23	52.29±5.40
	AtOrC (CNN,M=10)	73.39±4.96	54.28±3.83	76.19±3.86	86.00±1.28	62.99±6.72	43.89±1.68
	AtOrC (INCEP,M=19)	75.80±4.69	64.00±4.08	80.29±2.64	**91.00±1.37**	**73.59±4.17**	57.49±5.98
	AtOrC (CNN,M=19)	**76.79±4.14**	**64.57±4.98**	**80.39±2.53**	90.91±1.22	72.39±3.38	**58.49±4.96**

with nodes less than *max_node*, the matrix size is increased by repeatedly appending the rows of the original graph matrix in circular manner until the input matrix size becomes equal to *max_node* size. The *max_node* values of corresponding datasets are given in table 5.1.

Architecture details of Classification Module

- **CNN:** The convolution neural network used for classification (blue color blocks in figure 5.2) has the following architecture: a CNN layer with kernel size of 5x1, followed by another CNN layer with kernel size of 3x1. We have used a *valid* convolution operation for both CNN layers i.e. we didn't use any padding. The output is flattened and given to dense layer with 16 neurons followed by another dense layer with neurons equal to number of classes. The final layer is of softmax layer. The cross-entropy loss function is used with $l2$ regularizer on both CNN and dense layers.

- **Inception:** Instead of using a specific kernel size, it is proved that ensemble of output from multiple kernels performs better. Hence, inspired by the InceptionNet [2], we have used inception module instead of CNN layer. One inception module has two branches. One branch performs convolution with 7x2 kernel and other branch has two layers of CNN, first with 5x2 kernel followed by 3x1 kernel. As the output from two branches has the same size, it is concatenated and 1x1 kernel is applied to reduce the number of kernels to 4. Figure 5.5 shows the neural network architecture of Inception module. We have used two layers of such inception module. In table 5.2, the result with name AtOrC (CNN,M=10) and AtOrC (CNN,M=19) shows the result of applying CNN and AtOrC (INCEP,M=10) and AtOrC (INCEP,M=19) shows the result of applying inception module for graph classification.

The only hyperparameter for the AtOrC is a glimpse size denoted by M. It is the number of nodes to look at for graph classification. For the experiment, we have used two values $M = 10$ and $M = 19$. Other hyperparameters we optimized are batch size, learning rate. We have used Adam optimizer [88] for mini-batch gradient descent algorithm. The code is written in TensorFlow [89]. All experiments were run on a machine with GEFORCE GTX1080 Ti GPU with 12GB GPU memory.

5.4.3 Results

Comparison of different Graph classification algorithms on various datasets is reported in table 5.2. In almost all the datasets, AtOrC performs better than the existing Graph Kernel and Deep Learning based approaches. In case of PTC and MUTAG, performance of AtOrC is in comparison with existing state-of-the-art, even when the the AtOrC is using very less number of nodes than number of nodes used by other algorithms for performing graph classification.

It is interesting to see that just by attending to right set of nodes and ordering them properly, even with only glimpse of 10 nodes and CNN, IMDB-M is able to achieve accuracy of 52.29, outperforming the state-of-the-art algorithms by 4.46% . We could get best classification accuracy of 58.49 with improvement of 10.66% on the same IMDB-M dataset. For PROTEINS, D&D and IMDB-B dataset, we could achieve improvement of 0.9%, 3.12%, 3.56% respectively.

5.5 Discussion

5.5.1 Using AtOrC for Explaining Network Decisions:

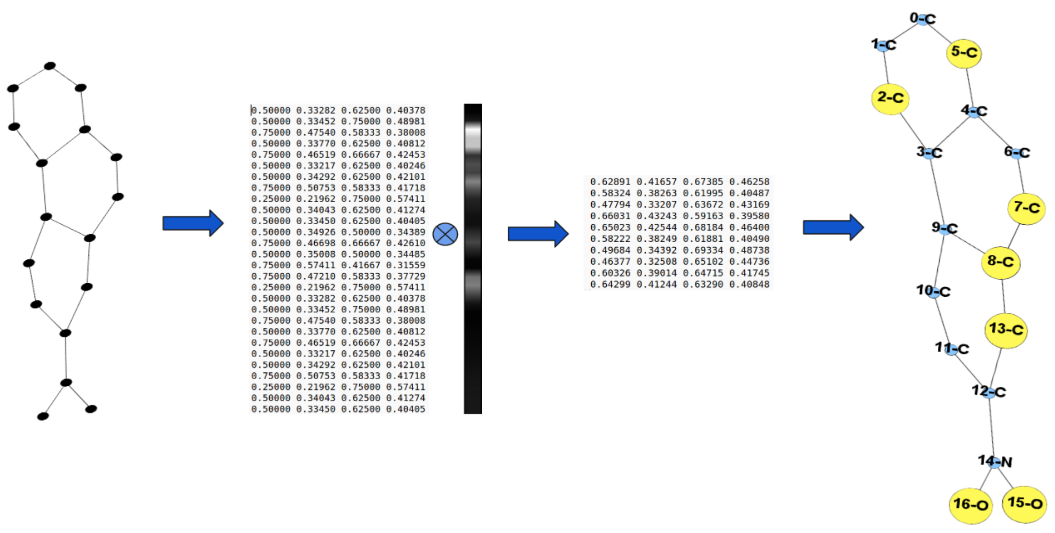
The SoftAttention module learns 1D Gaussian attention maps to select the few nodes for further analysis. Figure 5.7 and 5.8 shows the attention map generated by SoftAttention for one input graph of bioinformatics and social network datasets. The glimpse size is set to 19, that means, the SoftAttention will learn 19 1D Gaussian filters, whose 19 means and variances are learned by the backpropagation. As we know which rows are being attended by the Gaussian filters, we can use those for understanding the decisions taken by the AtOrC .

Figure 5.1 shows attention map generated for a graph in IMDB-MULTI dataset and it explains how attention maps are useful in understanding the decision of the algorithm. In the forward pass, the entire graph is given to the SoftAttention module and it learns the attention map that selects only few nodes. For understanding the decisions, we can trace back to the nodes being attended and verify whether the neural network is actually looking at the right set of nodes. In the attention map, white color region indicates that nodes corresponding to those rows have major contribution in graph classification. Gray color regions indicate that those nodes have partial contribution and black color region tells that those nodes are not considered for the classification task. Figure 5.6 shows the attended nodes of a graph from MUTAG dataset.

5.5.2 Ablation Study

SoftAttention: *Effect of glimpse size*
The real-world graphs could be very huge. Hence, considering all nodes in the graph for the analysis is not feasible and also not scalable. One way to overcome this problem is to consider only subset of nodes. Existing algorithms use predetermined criteria to select the nodes such as using Node Degree, PageRank or Weisfeiler-Lehman node labeling algorithm. There are three major drawbacks of using such criteria: i) It requires precomputing of labels which will be used later for selection, leading to extra overhead. ii) This process is not differentiable, and hence not easily pluggable into neural network architecture. iii) As we don't know which nodes are important beforehand, the only way is to try different selection algorithm and choose the one that performs best relative to others. We propose a novel solution of SoftAttention module that is designed for "learning to select" by attending to subset of nodes.

By attending to the right set of nodes, SoftAttention module was able to help in performing better at graph classification. To see the effect of attention size on the performance, we varied the



```
0.50000 0.33282 0.62500 0.40378
0.50000 0.33452 0.75000 0.48981
0.75000 0.47540 0.58333 0.38008
0.50000 0.33770 0.62500 0.40812
0.75000 0.46519 0.66667 0.42453
0.50000 0.33217 0.62500 0.40246
0.50000 0.34292 0.62500 0.42101
0.75000 0.50753 0.58333 0.41718
0.25000 0.21962 0.75000 0.57411
0.50000 0.34043 0.62500 0.41274
0.50000 0.33450 0.62500 0.40405
0.50000 0.34926 0.50000 0.34389
0.75000 0.46698 0.66667 0.42610
0.50000 0.35008 0.50000 0.34485
0.75000 0.57411 0.41667 0.31559
0.75000 0.47210 0.58333 0.37729
0.25000 0.21962 0.75000 0.57411
0.50000 0.33282 0.62500 0.40378
0.50000 0.33452 0.75000 0.48981
0.75000 0.47540 0.58333 0.38008
0.50000 0.33770 0.62500 0.40812
0.75000 0.46519 0.66667 0.42453
0.50000 0.33217 0.62500 0.40246
0.50000 0.34292 0.62500 0.42101
0.75000 0.50753 0.58333 0.41718
0.25000 0.21962 0.75000 0.57411
0.50000 0.34043 0.62500 0.41274
0.50000 0.33450 0.62500 0.40405
```

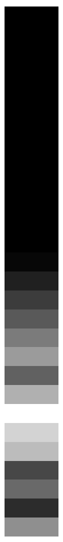
```
0.62891 0.41657 0.67385 0.46258
0.58324 0.38263 0.61995 0.40487
0.47794 0.33207 0.63672 0.43169
0.66031 0.43243 0.59163 0.39580
0.65023 0.42544 0.68184 0.46400
0.58222 0.38249 0.61881 0.40490
0.49684 0.34392 0.69334 0.48738
0.46377 0.32508 0.65102 0.44736
0.60326 0.39014 0.64715 0.41745
0.64299 0.41244 0.63290 0.40848
```

Figure 5.6: The figure shows a graph from MUTAG dataset. We can interpret the attended nodes of this graph based on the attention map generated by the SoftAttention module. The node labels indicate the node id followed by the atom of the molecule-C: Carbon atom, O: Oxygen atom and N: Nitrogen atom. The attended nodes are highlighted by bigger yellow color nodes in the rightmost graph. The proposed algorithm selected only five Carbon atoms and two Oxygen atoms to classify whether the graph is mutagenic.

glimpse size from 10 nodes to 20 nodes with an interval of 2. For fair comparison, we removed the SoftOrdering module, so that we can see only the effect of SoftAttention on classification accuracy. Figure 5.9 shows the line plot of glimpse size vs the classification accuracy. For the experiment, we chose two bioinformatics datasets: MUTAG, PROTEINS and two social network datasets: IMDB-BINARY, IMDB-MULTI. The light colored region around the line indicates the standard deviation. As it is clear from the plot that, as we increase the glimpse size, AtOrC performs better at a given task. For IMDB-BINARY and IMDB-MULTI, the accuracy becomes almost constant after glimpse size of 16, unlike MUTAG whose accuracy tends to increase even after glimpse size of 18. The classification accuracy of PROTEINS is consistent across all glimpse sizes.

SoftOrdering: *Predetermined ordering vs Learned ordering*
The Convolutional Neural Network can succeed in learning good features only when the nodes of all input graphs have same ordering. However, using existing centrality measuring algorithm restricts us to few ways of ordering nodes. Hence, we proposed a SoftOrdering module, a neural network that will learn the best ordering strategy all by itself. To test usefulness of the learned ordering, we removed SoftOrdering module. We used "Node Degree" as criteria to select the nodes and order them according to decreasing order of node degree. As node degree is already used for selecting and ordering nodes, we also removed the SoftAttention module. So the algorithm used for this ablation study does not have SoftAttention as well as SoftOrdering module, but only classification module. We selected 19 nodes and trained Convolutional Neural Network and Inception Network with architecture same as described in subsection 5.4.2.

Table 5.3 below shows the best classification accuracy of network that uses predetermined ordering and the proposed framework on 4 datasets: MUTAG, PROTEINS, IMDB-BINARY, and IMDB-MULTI. Average improvement of 16% verifies that the "learning to select" and "learning to


(a)Attention map for MUTAG: Gaussian filters select 19 nodes out of 28 by attending to different rows of the input matrix.

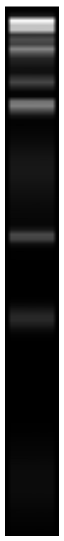
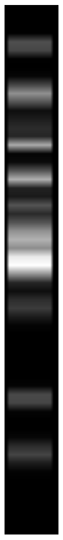

(b)Attention map for PROTEINS: Selecting 19 nodes out of 620 nodes.

Figure 5.7: Attention map generated by the SoftAttention module for Bioinformatics datasets with glimpse size of 19


(a)Attention map for IMDB-BINARY: Selecting 19 nodes out of 136 nodes.

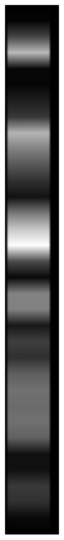

(b)Attention map for IMDB-MULTI that selects 19 nodes out of 89 nodes.

Figure 5.8: Attention map generated by the SoftAttention module for Social Network dataset. Here also the glimpse size is set to 19

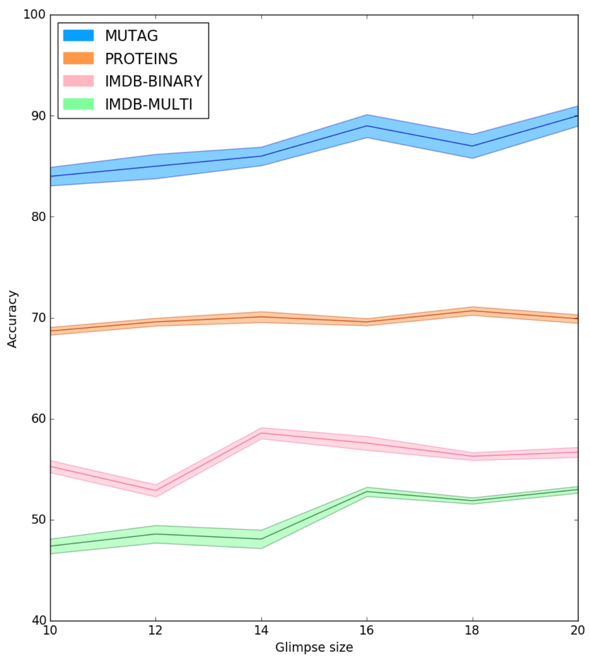


Figure 5.9: Line plot showing the effect of increasing the glimpse size on the classification accuracy when only SoftAttention module is used.

order", which are performed by the SoftAttention and SoftOrdering module, both play an important role in performing better at classification.

Table 5.3: Comparison of proposed algorithm, that learns to select the nodes and order them, with predetermined node selection and ordering strategy. Here we have used node degree as criteria for node selection and ordering

Datasets	**Node degree**	**AtOrC**
PROTEINS	68.49±2.53	76.79±4.14
MUTAG	65.26±8.55	91.00±1.37
IMDB-B	51.09±4.06	73.59±4.17
IMDB-M	49.69±6.92	58.49±4.96

Importance of Using Pretrained Weights in the SoftOrdering Module: To verify whether using the pretrained weights in SoftOrdering module helps in achieving better accuracy at graph classification task, we compared the the results against the neural network initialized with Glorot [90] weight initialization. We performed hypothesis test using two-tailed student's t-test with null hypothesis H_0 :*the weight initialization does not have any effect* and alternate hypothesis H_a :*the weight initialization has an effect*. The experiment has shown that the use of pretrained weights in SoftOrdering module has statistically significant performance difference on graph classification task with confidence level of 95%. The Table 5.4 shows the result of graph classification as well as the hypothesis testing.

Table 5.4: Table shows comparison of classification accuracy of AtOrC when weights in SoftOrdering module are initialized with (i)Glorot initialization and (ii)pretrained weights. Results in bold are indicate statistically significant performance difference under null hypothesis test with confidence level of 95% ($p = 0.05$)

Datasets	**Glorot Initialization [90]**	**Pretrained Weights**
MUTAG (CNN)	90.00±1.09	**90.91±1.22**
MUTAG (INCEP)	87.00±1.09	**91.00±1.37**
PROTEINS (CNN)	70.89±5.10	**76.79±4.14**
PROTEINS (INCEP)	72.49±4.05	**75.80±4.69**
PTC (CNN)	50.00±5.47	64.57±4.98
PTC (INCEP)	56.50±3.90	**64.00±4.08**
IMDB-B (CNN)	66.19±4.66	**72.39±3.38**
IMDB-B (INCEP)	67.89±4.88	**73.59±4.17**
IMDB-M (CNN)	53.79±7.38	58.49±4.96
IMDB-M (INCEP)	49.19±7.99	**57.49±5.98**

5.6 Conclusion

In this chapter, we proposed Attend, Order and Classify, a novel end-to-end deep neural network for graph classification. The framework is differentiable and does not need any preprocessing on the input graph. The framework is comprised of the SoftAttention Module that learns which nodes to select for further processing and the SoftOrdering Module learns in which order the nodes should be aligned so that the convolution neural network can learn the local features easily and perform better at graph classification task. We validated the effectiveness of the AtOrC by performing extensive experiments on 6 standard benchmark datasets. Comparison with 8 baselines, shows that the Attend, Order and Classifyis highly competitive with state-of-the-art algorithms. An important advantage of AtOrC is its ability to explain the decisions made by the network.

Chapter 6

Conclusions

The main focus of this dissertation is to analyze graph structured data from deep learning perspective. To this end, we, here, summarize our contributions on three graph analysis tasks.

Anomaly detection in edge attributed graphs:

Plethora of work has been done in finding anomalies in graphs. Some of the work consider only structure of the graph, other consider node attributes along with graph structure. Consider the DBLP graph where nodes are authors and two nodes share an edge if the two authors have collaborated in publishing a paper. If two authors work in different domains, say Medicine and Computer Vision with paper published in Data Mining conference. This collaboration is very unique and hence must be marked as an outlier. However, if we use just consider the graph structure or node attributes is not sufficient. We have to consider the details of paper published, which is nothing but the details of the edge connecting the two authors. Hence, to find such peculiar outliers, we proposed novel algorithm *HCODA:* Holistic Community-based Outlier Detection Algorithm in Chapter 3. In this framework, we consider graph structure, node attributes as well as the edge attributes. We model the problem using Hidden Markov Random Field over the communities in graph and provide inference using Expectation-Maximization algorithm. Experiments on synthetic dataset and real-world dataset validate the effectiveness of the proposed algorithm.

Representation learning on temporal graphs:

Most of the algorithms proposed on analyzing the temporal graphs are designed to do only a specific pre-determined task. However, if we wish to solve a new problem on temporal graphs, we have to develop a new algorithm from scratch. This motivated us to design an algorithm that learns the representation on the temporal graph in such a way that, it is independent of the downstream applications. We proposed an unsupervised algorithm, STWalk in Chapter 4, that considers the neighbors of a node in current time-step graph as well as past time-step graphs. It performs random walk on the spatio-temporal graph. The framework uses Skipgram algorithm for learning the representation of node trajectories. The trajectories which are closely related in spatio-temporal graph, their representations are near-by in embedding space. We have shown that the learned trajectory representations are not only useful in classification but also in finding the drift-point in the temporal graphs. We also demonstrated that arithmetic operations on these trajectory representations yield interesting and interpretable results.

End-to-end framework for graph classification:

Our final contribution in understanding graph data is about extending the deep learning concepts to graph structured data. Despite rapid progress in deep learning, there have very few attempts in using it to solve graph based problems. The major reasons that restricts the direct use of CNN on graphs are i) variable number of nodes to process and ii) no unique way of ordering the nodes. Hence, we proposed AtOrC in Chapter 5, a novel end to end deep neural network framework that "learns to select" the important nodes as well as "learns to order" them, all in a single pass of the input graph. the algorithm was able to outperform state of the art method with 10.66% improvement in classification accuracy on IMDB-MULTI dataset. With increasing importance of understanding the decision made by the deep neural network, we also presented an novel way of interpreting the decision made by the AtOrC by using the attention map generated by the SoftAttention module. To the best of our knowledge, our proposed framework is the first effort in interpreting the neural network decisions of graph based algorithm.

Future directions:

One promising future direction for graph analysis is to borrow concepts from Physics. The field of "Spring-Mass Systems In Equilibrium" studies the network of masses connected by springs and its behavior under equilibrium condition. This systems is very similar to graphs we study in social networks. Consider the problem of link prediction, where the algorithm needs to suggest a most plausible edge to be added between two nodes. This problem can be formulated as, adding a spring between two masses such that it does not disturb the equilibrium condition i.e. the new connection should have the least effect on equilibrium condition.

Despite significant advances in deep learning, one central challenge in graph analysis lies in how we can design an architecture that works directly on graph structure. The techniques used today for graph analysis is either the Convolutional Neural Network or Recurrent Neural Network and these methods are deigned by keeping in mind the specific structure of the data on which they will be used on. For example, the CNNs are designed for images or in general, fixed size data that can be represented in regular grid like structure. The RNNs are designed by keeping mind the text or any sequential data with varying length. Now, if we consider the graph data, it neither has the regular grid like structure nor the sequential structure. Hence, existing graph analysis methods, try to modify the input data so that they can use existing deep learning algorithms. Instead, our effort should be devoted in finding the techniques specifically designed for graphs that exploits the random structure of input graph.

To make further progress, it is these kinds of problems that we must take closer look at.

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich et al. Going deeper with convolutions. Cvpr, 2015 .

[3] C. C. Aggarwal. Outlier Analysis. Springer, 2013.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A Survey. *ACM computing surveys (CSUR)* 41, (2009) 15.

[5] V. J. Hodge and J. Austin. A Survey of Outlier Detection Methodologies. *Artificial intelligence review* 22, (2004) 85–126.

[6] L. Akoglu, H. Tong, and D. Koutra. Graph based Anomaly Detection and Description: A Survey. *Data Mining and Knowledge Discovery* 29, (2015) 626–688.

[7] M. Gupta, J. Gao, C. Aggarwal, and J. Han. Outlier Detection for Temporal Data. *Synthesis Lectures on Data Mining and Knowledge Discovery* 5, (2014) 1–129.

[8] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood Formation and Anomaly Detection in Bipartite Graphs. In Fifth IEEE International Conference on Data Mining (ICDM'05). IEEE, 2005 418–425.

[9] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han. On Community Outliers and their Efficient Detection in Information Networks. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010 813–822.

[10] M. Gupta, J. Gao, and J. Han. Community Distribution Outlier Detection in Heterogeneous Information Networks. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2013 557–573.

[11] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova. Anomaly Detection in Dynamic Detworks: A Survey. *WIREs Comp Stat* 7, (2015) 223–247.

[12] L. Akoglu and C. Faloutsos. Event detection in time series of mobile communication graphs. In Army Science Conference. 2010 77–79.

[13] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2006 374–383.

[14] J. Abello, T. Eliassi-Rad, and N. Devanur. Detecting novel discrepancies in communication networks. In Data Mining (ICDM), 2010 IEEE 10th International Conference on. IEEE, 2010 8–17.

[15] M. Gupta, J. Gao, Y. Sun, and J. Han. Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers. In Proc. of the 18^{th} ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD). 2012 859–867.

[16] M. Gupta, J. Gao, Y. Sun, and J. Han. Community Trend Outlier Detection using Soft Temporal Pattern Mining. In Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases (ECML PKDD). 2012 692–708.

[17] G.-J. Qi, C. C. Aggarwal, and T. Huang. Community Detection with Edge Content in Social Media Networks. In 2012 IEEE 28th International Conference on Data Engineering. IEEE, 2012 534–545.

[18] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In Proceedings of the 22nd international conference on World Wide Web. ACM, 2013 119–130.

[19] B. Boden, S. Günnemann, H. Hoffmann, and T. Seidl. Mining Coherent Subgraphs in Multilayer Graphs with Edge Labels. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012 1258–1266.

[20] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han. Top-k interesting subgraph discovery in information networks. In 2014 IEEE 30th International Conference on Data Engineering. IEEE, 2014 820–831.

[21] C. C. Aggarwal and N. Li. On node classification in dynamic content-based networks. In Proceedings of the 2011 SIAM International Conference on Data Mining. SIAM, 2011 355–366.

[22] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. *arXiv preprint arXiv:1706.02216* .

[23] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu. Attributed Network Embedding for Learning in a Dynamic Environment. *arXiv preprint arXiv:1706.01860* .

[24] P. Sarkar, D. Chakrabarti, and M. Jordan. Nonparametric link prediction in dynamic networks. *arXiv preprint arXiv:1206.6394* .

[25] L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multi-mode networks. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008 677–685.

[26] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)* 47, (2014) 10.

[27] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online Learning of Social Representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14. ACM, New York, NY, USA, 2014 701–710.

[28] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016 855–864.

[29] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web. ACM, 2015 1067–1077.

[30] 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. ACM, 2015 891–900.

[31] S. Cao, W. Lu, and Q. Xu. Deep Neural Networks for Learning Graph Representations. In AAAI. 2016 1145–1152.

[32] 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. ACM, 2015 119–128.

[33] P. Goyal and E. Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *arXiv preprint arXiv:1705.02801* .

[34] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In Proceedings of the IEEE international conference on computer vision workshops. 2015 37–45.

[35] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In Advances in Neural Information Processing Systems. 2016 3189–3197.

[36] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In Proc. CVPR, volume 1. 2017 3.

[37] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph Attention Networks. *stat* 1050, (2017) 20.

[38] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In Advances in Neural Information Processing Systems. 2016 1993–2001.

[39] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In Data Mining, Fifth IEEE International Conference on. IEEE, 2005 8–pp.

[40] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In Learning Theory and Kernel Machines, 129–143. Springer, 2003.

[41] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In Artificial Intelligence and Statistics. 2009 488–495.

[42] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, (2011) 2539–2561.

[43] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In International conference on machine learning. 2016 2014–2023.

[44] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928* .

[45] P. Yanardag and S. Vishwanathan. Deep graph kernels. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015 1365–1374.

[46] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An End-to-End Deep Learning Architecture for Graph Classification .

[47] D. Chakrabarti. Autopart: Parameter-free Graph Partitioning and Outlier Detection. In European Conference on Principles of Data Mining and Knowledge Discovery. Springer, 2004 112–124.

[48] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting Anomalies in Weighted Graphs. In Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2010 410–421.

[49] B. Pincombe. Anomaly Detection in Time Series of Graphs using ARMA Processes. *ASOR Bulletin* 24, (2005) 2–10.

[50] T. Idé and H. Kashima. Eigenspace-based Anomaly Detection in Computer Systems. In Proc. of the 10^{th} ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD). 2004 440–449.

[51] C. C. Aggarwal, Y. Zhao, and P. S. Yu. Outlier Detection in Graph Streams. In Proc. of the 27^{th} Intl. Conf. on Data Engineering (ICDE). IEEE Computer Society, 2011 399–409.

[52] J. Besag. On the Statistical Analysis of Dirty Pictures. *Journal of the Royal Statistical Society. Series B (Methodological)* 259–302.

[53] F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports* 533, (2013) 95–142.

[54] L. Getoor and C. P. Diehl. Link mining: a survey. *Acm Sigkdd Explorations Newsletter* 7, (2005) 3–12.

[55] C. C. Aggarwal. An introduction to social network data analytics. *Social network data analytics* 1–15.

[56] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29, (2015) 626–688.

[57] W. L. Hamilton, R. Ying, and J. Leskovec. Representation Learning on Graphs: Methods and Applications. *arXiv preprint arXiv:1709.05584* .

[58] J. Tang, H. Gao, and H. Liu. mTrust: Discerning multi-faceted trust in a connected world. In Proceedings of the fifth ACM international conference on Web search and data mining. ACM, 2012 93–102.

[59] J. Tang, H. Gao, H. Liu, and A. Das Sarma. eTrust: Understanding trust evolution in an online world 2012.

[60] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised Learning of Video Representations using LSTMs. In ICML. 2015 843–852.

[61] F. Chollet et al. Keras. `https://github.com/fchollet/keras` 2015.

[62] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems. 2013 3111–3119.

[63] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on, volume 2. IEEE, 2005 729–734.

[64] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks* 20, (2009) 61–80.

[65] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, (2013) 83–98.

[66] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems. 2016 3844–3852.

[67] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* .

[68] S. Gold, A. Rangarajan et al. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks* 2, (1996) 381–399.

[69] R. S. Cruz, B. Fernando, A. Cherian, and S. Gould. DeepPermNet: Visual Permutation Learning. *arXiv preprint arXiv:1704.02729* .

[70] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

[71] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007* .

[72] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* .

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems. 2017 6000–6010.

[74] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. IEEE, 2015 3156–3164.

[75] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning. 2015 2048–2057.

[76] H. Xu and K. Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In European Conference on Computer Vision. Springer, 2016 451–466.

[77] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv preprint arXiv:1606.01847* .

[78] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. Learning to reason: End-to-end module networks for visual question answering. *CoRR, abs/1704.05526* 3.

[79] Z. Yu, J. Yu, C. Xiang, J. Fan, and D. Tao. Beyond bilinear: Generalized multimodal factorized high-order pooling for visual question answering. *IEEE Transactions on Neural Networks and Learning Systems* .

[80] D. A. Hudson and C. D. Manning. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067* .

[81] Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick, and A. van den Hengel. Visual question answering: A survey of methods and datasets. *Computer Vision and Image Understanding* 163, (2017) 21–40.

[82] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623* .

[83] V. Mnih, N. Heess, A. Graves et al. Recurrent models of visual attention. In Advances in neural information processing systems. 2014 2204–2212.

[84] T. Marwah, G. Mittal, and V. N. Balasubramanian. Attentive Semantic Video Generation using Captions. In 2017 IEEE International Conference on Computer Vision (ICCV). IEEE, 2017 1435–1443.

[85] G. Mittal, T. Marwah, and V. N. Balasubramanian. Sync-DRAW: Automatic Video Generation using Deep Recurrent Attentive Architectures. In Proceedings of the 2017 ACM on Multimedia Conference. ACM, 2017 1096–1104.

[86] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics* 35, (1964) 876–879.

[87] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics* 21, (1967) 343–348.

[88] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

[89] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al. TensorFlow: A System for Large-Scale Machine Learning. In OSDI, volume 16. 2016 265–283.

[90] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010 249–256.