MULTI-FACET GRAPH MINING WITH CONTEXTUALIZED PROJECTIONS

BY

CARL YANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Jiawei Han, Chair
Professor Jian Peng
Professor ChengXiang Zhai
Professor Jure Leskovec, Stanford University

ABSTRACT

The goal of my doctoral research is to develop a new generation of graph mining techniques, centered around my proposed idea of multi-facet contextualized projections, for more systematic, flexible, and scalable knowledge discovery around massive, complex, and noisy real-world context-rich networks across various domains. Traditional graph theories largely overlook network contexts, whereas state-of-the-art graph mining algorithms simply regard them as associative attributes and brutally employ machine learning models developed in individual domains (*e.g.*, convolutional neural networks in computer vision, recurrent neural networks in natural language processing) to handle them jointly. As such, essentially different contexts (*e.g.*, temporal, spatial, textual, visual) are mixed up in a messy, unstable, and uninterpretable way, while the correlations between graph topologies and contexts remain a mystery, which further renders the development of real-world mining systems less principled and ineffective. To overcome such barriers, my research harnesses the power of multi-facet context modeling and focuses on the principle of contextualized projections, which provides generic but subtle solutions to knowledge discovery over graphs with the mixtures of various semantic contexts.

To consolidate the power of contextualized projections, my research systematically studies the *multi-facet graph organization, modeling and applications* in depth– NEP [1] and SetEvolve [2] enable the construction of multi-facet graphs from heterogeneous networks and free text corpora (among other works [3, 4, 5, 6, 7]); TaxoGAN [8] and CondGen [9] study unique graph mining techniques under contextualized projections by learning the embedding of contextualized graph proximities and the generation of contextualized graph structures (among other works [10, 11, 12, 13]); MultiSage [14] and ClusChurn [15] demonstrate the applications of contextualized projections to real-world web recommendation and user modeling systems (among other works [16, 17, 18, 19, 20]). In contrast to existing works on network and graph mining, the pieces of my doctoral research together constitute a general and integral pipeline for comprehensive and systematic leverage of rich contexts in networks. Therefore, the development of contextualized projection is distinct from, but also complementary to various network mining paradigms, which has been broadly recognized by the research community and readily applied to industrial platforms across various domains.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Born in an era of information explosion, we are inevitably inundated with vast amounts of data. Among them, network or graph data are ubiquitous and indispensable in a variety of high-impact data mining scenarios, due to its unique and generic modeling of interconnected objects. For example, users on social platforms are generating tons of profiles and links every day, gene-protein interactions are being recorded in increasing volumes enabled by modern biomedical devices, and sensor networks are growing rapidly as a derivation of the gigantic network of things. Unfortunately, networks generated in the real world are often *massive* (ranging from thousands to billions of nodes and links), *complex* (containing informative units with various contents and structures), and *noisy* (suffering from inaccurate or missing objects, attributes and relations), challenging the design and deployment of both effective and efficient knowledge discovery systems. While there exist various network models (*e.g.*, attributed networks, heterogeneous networks, multi-view networks, hyper-networks) and abundant mining algorithms (*e.g.*, spectral analysis, message passing systems, embedding methods), none of them systematically explores the rich *correlations between relational topologies and semantical contexts*, which as I demonstrate in my research, can be a key to the successful *organization, modeling and application* of real-world network data nowadays.

Among various types of skyrocketed data, networks or graphs are important but arguably the most difficult to handle, due to their lack of fixed structures (in comparison to grid-like images and sequence-based texts) and the well-known property of small world (which ultimately requires everything in memory for real-time access to even small local neighborhoods). Can real-world networks be organized and stored in a *multi-facet context-aware fashion* so that subsequent data mining models can be learned specifically in the most relevant and necessary context without wasteful global computations? Can we learn the *semantic-topology mappings* on context-rich networks to capture (and then predict or infer) the changing graph proximities or persisting graph structures under different semantic contexts? Can our network computational systems be *robust, flexible and scalable* to support various real-world industry-level applications? To answer these questions, my dissertation research outlines a systematic pipeline of *multi-facet graph mining* towards knowledge discovery over *context-rich* real-world networks, which consists of the following three steps.

1.1 MULTI-FACET GRAPH ORGANIZATION

Organizing massive networks with complex contexts into a multi-facet structure. Nowadays network data are skyrocketing in volume (*e.g.*, a publication network like DBLP has 4 million paper nodes, a single network like Facebook has 2 billion user nodes), making the execution of any single network model extremely hard, if not impossible (*e.g.*, to deploy a typical graph convolutional network for one billion nodes requires weeks of effort by teams of experienced research engineers). Moreover, nodes, links, and subnetworks can all be associated with complex multi-facet contexts (*e.g.*, user profiles can include numerical and categorical attributes as well as personalized free texts, commercial products are characterized by images, textual descriptions and reviews, entities in knowledge graphs can have various relations, gene-protein subnetworks can correspond to pathways of different diseases). *How to properly organize the massive network data w.r.t. complex contexts to facilitate subsequent data mining algorithms and applications?*

Organizing heterogeneous networks into multi-facet graphs. NEP [1] extends existing research on hierarchical topic modeling and assumes well-designed textual facets based on computable topic taxonomies. It focuses on the task of assigning nodes in heterogeneous networks with the correct topic labels, based on limited weak links between a small set of nodes on the networks and topics. Unlike existing network classification or embedding models, NEP generalizes linear label propagation on single-typed networks to nonlinear embedding propagation on heterogeneous networks, and further breaks free the requirements of domain-specific pre-defined semantic units called meta-paths. By leveraging a novel dynamically composed modular neural network trained with an efficient two-step uniform path sampling strategy, it achieves 5%-12% improvements on six datasets across various domains over the strongest baselines from the state-of-the-art on semi-supervised classification of network nodes with multi-facet labels with extremely scarce label data. Moreover, the training time of NEP is much less than most of the state-of-the-art baselines.

Organizing free texts into multi-facet graphs. Different from NEP, SetEvolve [2] studies the construction of networks from scratch from free text corpora. Moreover, besides the topic facet, it further considers the incorporation of various ordinal facets such as time, age and rating. Particularly, SetEvolve leverages a unified two-step framework based on nonparanomal graphical models to firstly identify entity sets from documents relevant to each topic, and then construct series of evolutionary networks along the ordinal facet. For example, on a CS literature corpus, SetEvolve flexibly generates series of concept networks evolving along the year dimension centered around topics like data mining and

computer vision, which can be used for subsequent tasks like knowledge summarization and pattern mining. The nonparanomal graphical model is leveraged for theoretically guaranteeing the accuracy of constructed networks when entities are modeled as variables in a complex dynamic system with ordinal discrete observations (*i.e.*, occurrence in documents), and empirically achieves 9%-21% improvements over baselines on synthetic data.

Other works on multi-facet graph organization. Besides NEP and SetEvolve, doc2graph [3] studies the end-to-end generation of concept maps from free texts under the weak supervision of document classification; ARP [4] studies the relation contexts on links, AutoPath [5] studies the long-range contexts on paths, whereas Nest [6] and Hinse [7] study the higher-order contexts on motifs and meta-graphs (*i.e.*, substructures in homogeneous and heterogeneous networks, respectively). Together, these pieces systematically pave the way towards the organization of high-quality and comprehensive multi-facet graphs, while still leaving many potentially interesting questions open, such as the joint learning of facet structure and graph projections.

1.2 MULTI-FACET GRAPH MODELING

Learning the correspondence between graph topology and semantic context. The successful organization of multi-facet graphs essentially leads to the context-aware decomposition of massive complex real-world networks into controllable smaller subnetworks with clear structures and semantics, which allows us to further design series of dedicated and simplified local models for the joint knowledge discovery in multiple correlated fine-grained subspaces. Subsequent to multi-facet graph organization, my research addresses the question: *How to effectively model and mine the the series of subnetworks organized with multi-facet contexts?*

Contextualized proximity embedding. Recent research on network mining has been largely propelled by the rapid development of embedding algorithms, which aims at transferring node proximity on networks into distributional vectors and useful to various downstream machine learning tasks. Through the unique modeling of multi-facet graphs, TaxoGAN [8] is proposed to model two novel but important properties, *i.e.*, contextualized node proximity and hierarchical label proximity, for the co-embedding of networks and associative context taxonomies. Particularly, TaxoGAN models subnetwork nodes and labels in each facet in an individual embedding space via a series of graph generative adversarial networks (GAN), and then relates and transfers proximities among the hierarchical label specific spaces by

stacking the GANs with learnable graph encoders. In this way, the abstract-to-fine network generation process is aligned with the context hierarchies, which significantly improves the overall network embedding (*e.g.*, 11%-38% gain on hierarchical node classification), and at the meantime enables novel insightful tasks like conditional network proximity search and fine-grained taxonomy label visualization.

Contextualized structure generation. Although recent theoretical analyses and empirical studies on network embedding have largely pushed forward the translation of discrete network structures into distributed representation vectors, they seldom consider the reverse direction of generating graphs with given semantic contexts. Consider the example of biomedical gene networks, CondGen [9] is proposed to leverage multi-facet graphs by learning the latent context-to-network correspondence from data-rich subgraphs of closely related diseases where observations are broadly available, and flexibly generating insightful novel graph structures where observations are scarce or totally missing, so as to improve the comprehensive understanding and prediction of disease development. Technically, CondGen addresses the inherent challenges of flexible context-structure conditioning and permutation-invariant graph generation faced by such contextualized graph generation through a powerful graph variational generative adversarial network model, and is validated on two deliberately created benchmark datasets of real-world contextualized TCGA gene networks and DBLP author networks.

Query-specific network construction. While networks are ubiquitously used to model real-world interconnected objects, typical network mining tasks are often done on particular query sets of objects, which does not require full access to and computation over the whole massive networks including all objects and interactions in the datasets. By leveraging network contexts such as object attributes that characterize the essential properties of objects, cube2net [10] is proposed to dynamically construct small and complete subnetworks that are the most relevant to the particular queries based on the multi-facet graph organization. A light-weight reinforcement learning algorithm is developed to efficiently search over the context embedding space and find the near optimal combination subnetworks corresponding to the query-specific semantics that can support various downstream mining tasks (*e.g.*, author classification on DBLP, place recommendation on Yelp) with empirically validated effectiveness (*i.e.*, 3%-13% improvements over baselines) and efficiency (*i.e.*, 63%-75% improvements over baselines).

1.3 MULTI-FACET GRAPH APPLICATION

Advancing real-world downstream tasks with multi-facet graph mining. The proposed multi-facet graph mining framework jointly organizes network topology and context and models their inherent correspondence, which is beneficial in various real-world network application scenarios. By leveraging the powerful multi-facet graph mining framework to conduct large-scale collective analysis, prediction and intervention over multiple mainstream industrial platforms, my research demonstrates a rich set of success examples to answer the question of *what real impacts can multi-facet graph mining bring to real-world industry-level applications.*

Web-scale recommendation with explicit graph contexts. Graph convolutional neural networks (GCN) have been intensively studied and shown effective in real-world recommender systems, but they ignore the various contexts under which objects interact, and are thus incapable of capturing the multi-facet object interaction patterns. MultiSage [14] is designed in Pinterest to model the interactions of pins given the explicit context provided by the boards which collect the pins. As a result, proper modeling of pin interactions under board contexts yields 9%-26% improvements over the existing production pipeline on general pin recommendation and enables novel contextualized pin recommendation. A distributed Hadoop2-based training pipeline is further developed to scale up MultiSage to the Pinterest production graph with 1.3B pins, 1.2B boards and 23B pin-board links (compared with most existing GCN models that only run on networks with thousands of nodes and links).

New user churn prediction with implicit graph contexts. Users quit the usage of online services (*i.e.*, churn) due to different reasons. Towards the rapid prediction and reaction to different types of user churn, ClusChurn [15] models the implicit context of user types by clustering users into interpretable groups based on their multi-facet daily activities (*e.g.*, chatting, content consumption, link formation) and then predicts users' churn rate based on their limited initial behavior data jointly with their latent user types through a scalable parallel LSTM with attention framework. Extensive data analysis and experiments on the Snapchat social networks show that ClusChurn provides valuable insight into user behaviors and achieves state-of-the-art churn prediction performance. The whole framework is deployed as a data analysis pipeline, delivering real-time analytical and prediction results to multiple relevant teams for business intelligence uses. The work has also been patented and covered by the official ACM Morning Papers on social media.

Other works on real-world applications of multi-facet graph mining. Through continuous collaboration with multiple industrial research labs, my research centered around

multi-facet graph mining has led to several other successful knowledge discovery engines that utilize the power of big data in a diverse spectrum of important real-world applications. ReLearn [16] (joint work with LinkedIn Economic Graph Research) infers the relationships underlying uniform friend connections in social networks; DeDup [17] (joint work with Facebook AI Research) de-duplicates place pages on the Facebook place network, Phine [18] (joint work with DiDi Big Data Lab) predicts user ratings on the DiDi transportation network, BLA [19] (joint work with Snap Research) infers user attributes and links on the Snapchat social network, and Pace [20] (towards the Yelp Data Challenge) performs location recommendation on the Yelp bipartite network. Framed in the flavor of principled integration of research advances and practical systems, many of them have been deployed in the collaborating corporations and attracted wide attention from both academic researchers and industrial practitioners.

Organization The remainder of this proposal is organized as follows.

- In Chapter 2, I provide a survey on network data mining, especially around context-rich heterogeneous networks that are close to the setting of my thesis.
- In Chapter 3, I present the overall overall framework of contextualized projections, based on a real-world demo system with a series of unique functions it enables.
- In Chapter 4 and Chapter 5, using real-world datasets, I give concrete in-depth examples of my doctoral research around the techniques and applications of contextualized projections.
- In Chapter 6, I conclude my doctoral thesis with a brief summarization of current accomplishments and future directions.

CHAPTER 2: LITERATURE REVIEW

In this chapter, I survey existing studies that are closely related to the novel multi-facet graph mining framework.

2.1 GRAPH MINING

Graph Embedding Earlier studies on graph (network) embedding are focused around the graph spectrum, which have genial theoretical supports but are hard to scale up due to the heavy computation of eigen decomposition [21, 22, 23, 24]. Recently, the successful Skip-gram model for word embedding [25] has motivated many proximity-based neural network embedding methods, which directly optimizes towards link and neighborhood preserving objectives [26, 27, 28] or random walk based proximities [29, 30, 31]. These methods are often unsupervised and only capture network link structures. As a natural integration of network node contents and link structures, graph convolutional neural networks (GCN) have been proposed and intensively studied nowadays [32]. Trained in a semi-supervised fashion, convolution-based models learn to jointly capture node contents, link structures, and labels in an end-to-end fashion, and is shown to yield state-of-the-art performance in various graph mining tasks [33, 34, 35]. Developed on simple homogeneous network settings, both proximity-preserving and convolution-based methods have attracted various follow-up works in more complex network settings such as multi-view networks, heterogeneous networks, and hyper-networks [36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65].

On the other hand, knowledge bases (KB) are a special type of graphs where nodes are connected by different relational links. Unlike both proximity-preserving methods and GCNs which are essentially enforcing the smoothness assumption on graphs by requiring nodes close in the graphs to have similar embedding vectors, in KB, node embeddings are optimized towards the translation consistency [66]. In particular, node and links are jointly embedded in the shared embedding space, where the facts in terms of triplets are preserved by requiring the two head nodes and the one connecting link to form a certain relation (*e.g.*, addition [67, 68, 69, 70, 71, 72, 73, 74], projection [75, 76, 77, 78, 79, 80, 81, 82, 83], learnable neural combination [84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94]). In real graphs, sticking to either smoothness-based or translation-based method may not be guaranteed to achieve the best task performance. Instead, they should be preferred or combined based on the recognition of the nature of links in general graphs and KBs.

Graph Generation Graph generation models have been studied for decades and widely used to generate synthetic graph (network) data with certain intrinsic structural properties, which are used towards the development and evaluation of various collective data analytical and mining assumptions and models [95, 96]. Earlier works on graph generation mainly use probabilistic graph models to generate graphs with particular properties such as random links [97], small diameters [98], power-law distribution [99] and preferential attachment [100]. They are manually designed based on sheer observations and prior assumptions.

Thanks to the surge of deep learning, many new graph generation models have been developed recently, which leverage different powerful neural network schemes in a learn-and-generate manner [101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111]. For example, GraphVAE [101] leverages a variational autoencode framework to learn the graph structures via graph convolutional neural networks (GCN) and generate new graphs by reconstructing the adjacency matrices from random samples; NetGAN [102] firstly converts graphs into biased random walks, then learns the generation of walks instead of graphs with generative adversarial networks (GAN), and finally assemble the generated walks into generated graphs; GraphRNN [107] regards the generation of graphs as node-and-edge addition sequences, and models it with a heuristic breadth-first-search node-ordering scheme and a hierarchical node-and-edge recurrent neural network (RNN). These neural network based models can often generate graphs with much richer properties and flexible structures learned on real-world graphs.

2.2 CONTEXT MINING

Heterogeneous Content Integration Nowadays real-world networks are ubiquitously associated with rich contents (*e.g.*, images in web networks [112], texts in publication networks [113], attributes in social networks [114, 115], *etc.*). Existing works on network embedding usually integrate such heterogeneous contents by employing domain-specific deep content encoders [116, 117]. For example, HNE [118] and SHNE [56] both devise various encoders for image and text contents on the web networks, which are used as the input of a first-order bilinear interaction function to further capture network structures; HetGNN [65] and GATNE [62] both leverage multiple type-aware GCN frameworks that naturally take different contents as inputs and project them into a shared embedding space. However, such integrations do not explicitly model the correlations between network contents and structures, and they do not consider the different interaction patterns among different contents.

Textual Content Mining Take textual contents as an example. Although domain-specific models like RNN [117] can be directly applied to convert text data into numerical vectors that can be further integrated with network mining models, text data themselves are rather complicated, and if studied in details, can provide much richer contexts than just numerical vectors.

Enriching networks with entity recognition and relation extraction. Recent research on text data mining has made great progress on turning unstructured texts into structured knowledge, through techniques like entity recognition [119, 120, 121] and relation extraction [122, 123, 124]. Particularly, entity recognition allows the retrieval of named semantic objects from unstructured free texts, whereas relation extraction studies the identification of pre-defined relations. Earlier methods for both tasks rely a lot on human defined patterns [125, 126, 127] and human annotated data [128, 129, 130], while recent advance on deep learning has shifted the research attention to weak supervision from distant label sources like knowledge base [131, 132, 133, 134] as well as powerful neural network language models [135, 136, 137]. The extracted objects and relations can then be naturally integrated into the existing networks as additional nodes and links [11].

Improving cube structures with automatic taxonomy construction. Besides nodes and links, in-depth text data mining also allows the automatic construction of taxonomies, which can naturally help flexible and data-driven selection of the cube structures, as the cube structures essentially consist of multi-facet taxonomies. Traditional methods mainly rely on lexical patterns to construct hypernym-hyponym taxonomies [138, 139, 140, 141], whereas recent ones leverage semantic word embedding and hierarchical clustering techniques to learn more flexible taxonomies [142, 143, 144, 145]. Depending on the taxonomy quality and end task, automatically constructed taxonomies from text mining can be directly applied as cube dimensions, or further tuned by supervised learning algorithms [146, 147] as well as human curation.

2.3 APPLICATION

My proposed CubeNet framework essentially adds a layer to the existing network data models. Therefore, the studies over it are mostly orthogonal and complementary to existing research on network data mining, and it potentially supports various network applications such as node classification, recommendation, and knowledge bases.

Node Classification One of the most representative tasks in traditional network data min-

ing is semi-supervised node classification, due to its wide applications in various domains [148, 149, 150, 151, 152, 114, 153]. Demonstrated in many recent works on network embedding, utility in node classification also highly reflects utility in other important tasks like graph classification, community detection, and link prediction [29, 26, 33, 154]. Therefore, we focus our discussion on node classification as the major traditional network mining application.

In the application perspective, as networks can be constructed in various ways, more accurate classification of nodes results in better performance of various tasks. For example, on the real-world social networks, homophily based smoothness regularization improves the classification of users [114, 115]; on the heterogeneous publication networks, meta-path based proximity improves the classification of papers [155, 156]; on the synthetic data adjacency networks, label propagation improves the classification of images [149, 152]. In CubeNet, node classification as well as other traditional network mining tasks can be modeled as they are in drilled-down subnetworks, where the mining algorithms largely remain the same but on specific small networks, so as to produce accurate results with minimum computations [11].

Recommendation Recommendation is essentially a link prediction task on the bipartite user-item networks. Recent research has shown the potential power of more complex context networks as supplementary sources for improving recommendation, especially in the cold start scenarios [157, 158]. For example, [159, 160] explore user interactions on social networks to improve item recommendation; [161, 162] leverage location proximities on geographical networks to improve place recommendation; [94, 163] leverage entity relations on knowledge graphs to jointly improve knowledge representation and item recommendation. However, their leverage of network information in recommendation is still limited, because they seldomly consider the different interaction contexts of nodes on the networks. CubeNet is proposed to fill this gap by subtly modeling the semantic contexts of node interactions, which provides an edge in general recommendation, and enables novel but important flexible contextualized recommendation.

CHAPTER 3: OVERVIEW

Nowadays, due to the advances in digital devices, we have witnessed an explosion of network data, which are massive, complex and noisy. Traditional network models based on spectral graph theory focus on network structures while overlooking network contexts. Based on them, later general message passing systems uniformly incorporate network contexts as node features propagating among local neighborhoods, which are often linear and shallow. Recent popular graph embedding methods insert additional layers of domain-specific feature transformers (*e.g.*, deep neural networks) into the message passing systems to model more complex network contexts. However, contexts in real-world networks are often not only complex, but also multi-facet (*e.g.*, temporal, spatial, topical, visual) and multi-granular (*i.e.*, bearing hierarchical structures). Moreover, they constantly influence the corresponding network structures through complicated interactions among themselves. Without the realization and subtle modeling of the correspondence between graph topology and multi-facet semantic contexts, existing network models are incapable of discovering accurate, flexible and insightful knowledge. For example, consider the following context-specific questions like: *Where and when will a user make a link to another on a social platform? Which references will a data mining paper make on a machine learning problem? How does a gene-protein pathway look like for males at the age of 30-34 with a heart disease?* Such questions are common and important across various applications and domains. However, being simply the multi-facet contextualized version of the extensively studied network mining tasks like link prediction and structure learning, questions like them can hardly be systematically answered by existing network mining paradigms.

3.1 MAIN DESIGNS AND INNOVATIONS

To answer the questions above, my dissertation focuses on principled knowledge discovery over context-rich real-world networks. Particularly, I propose to project massive networks into controllable small subnetworks with clear semantics and structures, then in parallel mine the semantically correlated subnetworks with a series of properly regularized simplified local models, and finally output fine-grained knowledge optimized towards the information need of specific queries. I attempt to make the whole process systematic (*i.e.*, captures comprehensive multi-facet contexts), intelligent (*i.e.*, understands the essential correlations among contexts and structures), flexible (*i.e.*, focuses on query-specific substructures) and scalable (*i.e.*, works for web-scale network data).

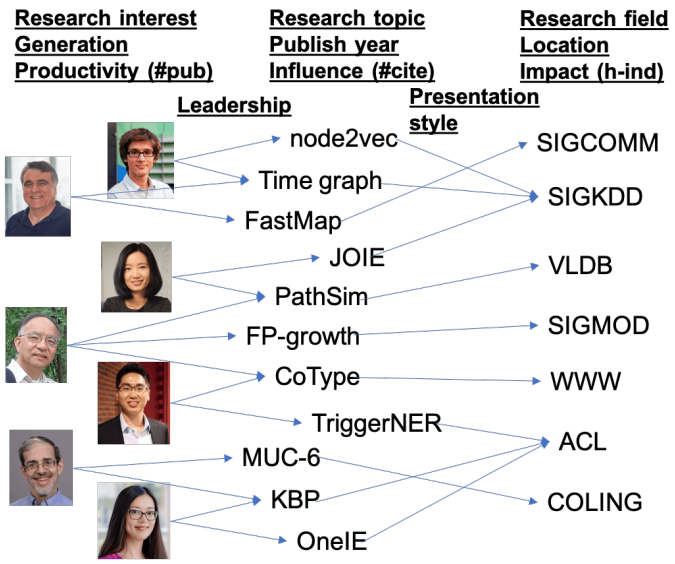


Figure 3.1: Toy example of a real-world context-rich network.

3.1.1 Context-Rich Networks: An Illustrative Example

Real-world networks nowadays are ubiquitously associated with various contexts. The contexts can originate from different data sources, and exist in different formats, such as *temporal*, *textual*, *visual*, and *spatial*. Take Figure 3.1 as an example, which is a toy yet realistic context-rich network constructed from DBLP. In particular, the network itself is heterogeneous [155], consisting of nodes with types such as *authors*, *papers*, and *venues*, as well as links with types such as *write* and *published in*. However, beyond types, both nodes and links are further associated with more detailed and semantic-rich contexts. For example, the *author nodes* can be described with *research interest*, *generation*, and *productivity*, whereas the *write links* have attributes such as leadership (differentiating leading authors and collaborative coauthors). We call such attributes and descriptors *contexts*, and networks with such contexts *context-rich networks*. We formally define the two concepts as follows.

Definition 1 *A network context is a map $c : x \to \mathbb{R}^{k^*}$, where x can be a node, a link, or a subgraph. $k^* = \sum_{i=1}^{d} k_i$, where d is the number of facet, and k_i is the dimension of the i-th facet. Naturally, in any particular network, d and k_i's are decided by $T(x)$, which denotes whether x is a node, a link, or a subgraph. and what type of node (link, subgraph) x is.*

Continue with the example in Figure 3.1. When x is an *author node*, *i.e.*, $T(x) =$*n_author*, the number of facets is three, *i.e.*, $d = 3$. Subsequently, k_1 and k_2 are the dimensions of the corresponding categorical attributes *research interest* and *generation*, both represented as one-hot vectors. $k_3 = 1$ is the dimension of a single value, *i.e.*, number of publications.

Likewise, when x is a *write link*, *i.e.*, $T(x) =$*l_write*, the number of facets is one, *i.e.*, $d = 1$. Subsequently, $k_1 = 1$ is the dimension of a single binary value, *i.e.*, leadership or not.

Definition 2 *A context-rich network is a network $G = \{V, E, T, C\}$, where V is the set of nodes, E the set of links, T the typing function, and C the set of network contexts. Note that, the contexts can be presented either partially only on nodes, links or subgraphs, or fully on all of them, and when the contexts are presented on any of them, e.g., nodes, they can be partially presented only on one type of nodes or fully on all types of nodes.*

Our definition of context-rich network is a natural extension of existing network data models like dynamic networks, attributed networks, heterogeneous networks, multi-view networks, and hypernetworks. It is rigorously defined, yet general and flexible towards the modeling of various real-world network data. In the following, we will demonstrate a unique powerful operation that fully leverages the design of context-rich networks.

3.1.2 Multi-Facet Network Mining with Contextualized Projections

Now we define the *contextualized projection* on context-rich networks. The idea is to *filter* the massive, complex, and noisy networks *w.r.t.* particular contexts in a multi-facet manner, so as to get sets of smaller networks supposedly with simple and clear semantics and structures, which are easier to model and more useful towards particular downstream data analysis and mining tasks/queries.

Definition 3 *A contextualized projection of a context-rich network is map $p : \mathcal{G} \to \mathcal{G}$, which computes the intersection of all subsets selected by a set of rules $\Theta_p = \{c_i(x) \in \bar{c}_i | c_i \in C\}_{i=1}^{r_p}$, where each r_p is the number of rules specified by p, each c_i is a network context associated with the original context-rich network G, and $\bar{c}_i$ is a set of constant values of c_i (can be infinite when $c_i(x)$ is continuous).*

For each rule $c_i(x) \in \bar{c}_i$, p first identifies the type of objects t_i that are associated with c_i, and then selects a subset $\{x | c_i(x) \in \bar{c}_i\}$ from the set $\{x | T(x) = t_i\}$. Therefore, each rules selects a subset of nodes, links or subgraphs. According to the commutation property of intersection, the results are unique given a fixed set of rules. Finally, post-processing like the removal of isolated nodes, incomplete links and small unconnected subgraphs can be further applied based on the need of downstream tasks. Note that, by default we claim a contextualized projection as valid, as long as the resulting projected network is nonempty, while in real-world applications, we may also define validation rules based on certain graph properties of the resulting projected network to refuse useless projections.

Continue with the example in Figure 3.1. We can apply a contextualized projection on the *author node* with rule *research interest=network mining*, which effectively filters out certain authors that do not study *network mining*. Since no rules are enforced on links, subgraphs or other types of nodes, most of them remain the same, while some of them can be removed like the *write links* with no authors. This particular projection allows us to focus on the network mining community, which is clearly specified by the context semantics and has rather simplified structures, where downstream tasks relevant only to this community are expected to be performed with better accuracy and efficiency.

Besides the above rule, the general operation of contextualized projection allows one to further compile rules to project on other types of nodes, links, and subgraphs, as well as flexibly combine multiple rules to generate subnetworks based on the need of real tasks. Some examples of the real-world use cases may include interest-based communities in social networks, disease-related pathways in protein networks, temporal-dynamic interactions in physical networks, privacy-constrained clusters in decentralized networks and so on.

As we motivated in the example already, real-world applications often encourage or require us to perform local computations on the precisely projected subnetworks due to their preferred structure simplicity and close semantic relevance. However, in the following, we propose to further study and model the relations among different local subnetworks and the global original network, so as to improve the capture and leverage of the two types of essential information underlying network data, *i.e.*, *proximity* and *structure*, under our novel framework of contextualized projections.

We now introduce the two types of essential information in network data, *i.e.*, *proximity* and *structure*. The uniqueness of network data, in comparison to other types of data, is to serve as a model of interactions or relations among individual nodes. Insightfully, we highlight that such interactions and relations captured by networks characterize the objects in them by two *orthogonal* properties, that are *proximities* and *structures*.

3.1.3 Contextualized Graph Proximities

Proximity information. Intuitively, the *proximity* between two nodes u and v in the network G is transductive and always relative to other nodes $V/(u, v)$ in G. There is no absolute proximity, and thus there is no way to compare the proximities among nodes in different networks. Now we give some examples of existing network models or algorithms that aim to capture such *proximity information* of nodes in a network, which we categorize into two groups.

Path-counting proximity models. Traditional proximity models are essentially based on counting the paths between each pair of nodes, thus capturing their relative proximities. Famous models include PPR [164], SimRank [165], and PathSim [155]. Among them, PPR implicitly counts the (normalized) number of paths between the source v and target u, and the more important paths between v and u, the higher proximity score is assigned to (u, v). SimRank computes a similar proximity score by paying more attention to the pairwise notion, and counts the paths from two nodes u and v to all intermediate nodes. Thus, the more paths connecting u and v to the same nodes, the higher proximity score is assigned to (u, v). PathSim extends the path computation onto heterogeneous networks with different types of nodes and links, and counts the normalized number of paths constrained by priorly defined meta-paths. Similarly, the more constrained paths connecting u and v, the higher proximity score is assigned to (u, v).

Note that, by computing the pair-wise path-counting proximities, all such methods are focusing on the relative proximities of nodes in a network, and they care less about the actual structures of local neighborhoods of individual nodes, which we will stress in the following when elaborating on *structure* information. Moreover, all path-counting proximities can be computed by iteratively multiplying a specifically designed commuting matrix, which is equivalent to the inversion of a corresponding stationary matrix [164, 165, 155].

Embedding-based proximity models. Recent research on network data has been focused on representation (embedding) learning. One popular group of node embedding models represented by [29, 30, 26, 31] are developed under the very similar intuition of path-counting proximity models. Particularly, they apply the Skipgram model [25] from word embedding to optimize an embedding space where nodes observed more in the same truncated random walk sequences [29, 30] or connected more to the same neighboring nodes [26, 31] are drawn closer in the embedding space. In fact, these models are implicitly factorizing a specifically designed commuting matrix [166, 167], which essentially captures the same relative proximities of nodes in a network as the path-counting models, but with the additional product of an explicitly optimized embedding space that arranges nodes *w.r.t.* their relative proximities captured by the paths. Note that, the proximities among nodes captured in the embedding space, although are absolutely fixed in a network, are transductive in their essence as not comparable across different networks.

Another line of recent research on network representation learning is represented by the work of GCN [32]. In general, these network models are called graph neural networks (GNN) [33, 34, 35], which perform message passing on networks with learnable nonlinear feature projections at each step. There have been several research studies showing the

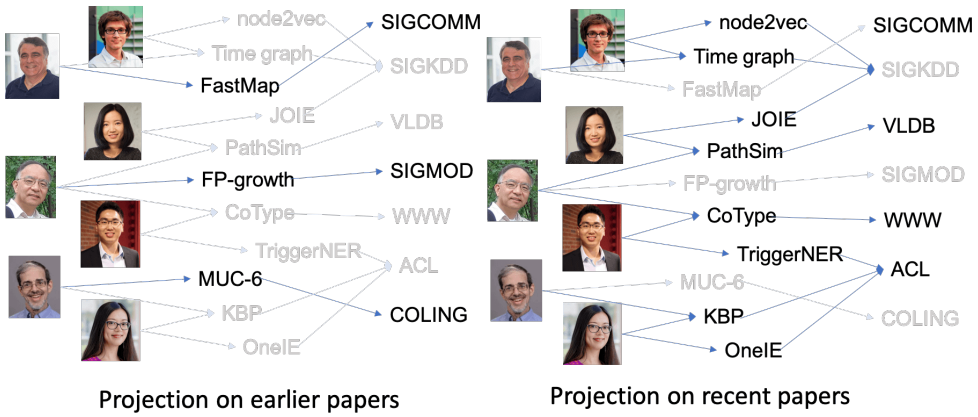


Figure 3.2: Networks projected regarding different paper-year contexts.

Context	Jure Leskovec	Christos Faloutsos	Yizhou Sun	Xiang Ren	Heng Ji	Ralph Grishman
venue-field: data	0.2928	0.9093	0.2253	0.0751	0.0587	0.0084
venue-field: ml	0.3235	0.0417	0.2474	0	0.2062	0.1875
venue-field: nlp	0.5926	0.1000	0.5217	0.1000	0.2880	0.2805
venue-field: ir	0.1381	0.8957	0.3975	0.0693	0.2266	0.1381
venue-field: bio	0	0	0.5714	0	0	0
paper-year: <2000	0	0.6342	0	0	0	0.0630
paper-year: 2000-2010	0.1799	0.8683	0.0849	0	0.0037	0.0036
paper-year: >2010	0.5123	0.9099	0.3972	0.1776	0.1643	0.0217
paper-cite: <5	0.2778	0.6330	0.2424	0.4231	0.1062	0
paper-cite: 5-20	0.2670	0.7544	0.3019	0.2541	0.2517	0.0531
paper:cite >20	0.2694	0.9008	0.2217	0.0397	0.0513	0.0187

Table 3.1: PathSim proximity (regarding meta-path A-P-V-P-A) between Jiawei Han and other authors on DBLP under different contextualized projections.

universal representation power of GNNs, which, as we will also stress in the following when elaborating on *structure* information, have shown GNNs to be mainly designed to capture the absolute *structure* of networks by approximating the WL network isomorphism test, since the multi-layers of nonlinear projections plus proper aggregation and readout mechanisms can universally approximate any functions in the spectral domain [168, 169, 170]. However, here we highlight that GNNs also implicitly capture *proximity* information. As reasoned in [20], when passing features in the network, GNNs implicitly ensure the smoothness among neighbors, *i.e.*, nodes closely connected in the network thus positionally close will have similar representations. Such smoothness is ensured by the aggregation mechanisms of GNNs, which usually combines the average representations of direct neighbors to that of the center node before any further projections. Therefore, no matter what projection functions are learned, the smoothness is ensured to some extent, so as the capturing of relative node *proximities*.

Contextualized graph proximity. Under different contexts, the relative proximities among nodes in networks can change. Continue with our toy example on DBLP. Figure 3.2 shows

the projections of the network in Figure 3.1 under different projections regarding paper-year. Clearly, the distances among authors change when different contextualized projections are applied. For example, some senior authors are weakly connected in the earlier years through few long paths, and they become more densely connected by many short paths as time goes by and their research interests converge.

In Table 3.1, we compute the PathSim similarity between Jiawei Han and the several other authors regarding meta-path A-P-V-P-A in the real-world DBLP dataset with a total of 26K authors, 177K papers, 45 venues and 529K links from 10 fields under several contextualized projections. In particular, this metric quantifies the peer-similarity between authors [155], which we present as an example from the many other metrics we have experimented with, to showcase the possibly different node proximity computed under different network contexts. The results are intuitive yet insightful. For example, regarding venue-field contexts, Jiawei Han is evaluated as a close peer to Christos Faloutsos in fields like data mining and information retrieval, because they are both the most well known researchers in those fields. However, although Jiawei Han and Yizhou Sun publishes a lot of papers together in data mining, they are not that similar because Jiawei Han is much more senior than Yizhou Sun. The two are more similar in fields like NLP and Bioinformatics, because they do also publish there, both with limited numbers of papers. Regarding paper-year contexts, Jiawei Han is significantly more similar to senior authors like Christos Faloutsos when papers in the earlier years are considered, and becomes more similar to junior authors like Jure Leskovec and Yizhou Sun as more recent papers are considered. Regarding paper-cite contexts, the similarity between Jiawei Han and other authors differs when papers with different numbers of citations are considered: most of the papers between Jiawei Han and Christos Faloutsos are better cited, while most of those between Jiawei Han and other authors are neither too unknown or too popular.

With the examples and analyses above, it is clear that context is important for accurate positional proximity and embedding in real-world networks. However, it is cumbersome to compute a single global model for the whole network or one local model for each projected network. Moreover, a single global model cannot highlight the contextualized proximity, while individual local models suffer from incomplete structural data and scarce labels with downstream tasks. Therefore, we study the joint and transfer learning of network embedding models, leading to successful models like TaxoGAN (Section 4.1-4.3).

3.1.4 Contextualized graph structure

Structural information. In contrast to *proximity* information which is always relatively measured in one network (thus transductive in nature), *structure* information is absolute and can be compared across different networks (thus inductive in nature). However, the term *structure* is still meaningless without the consideration of particular networks, because one node has no interesting structure by itself. Thus, the *structure* information of a node is always specified by its surrounding subnetwork, such as its direct or higher-order ego networks. The *structure* of a node v is absolutely fixed once its surrounding subnetwork $S(v)$ is specified, no matter what ambient network G the node v belongs to. Now we give some examples of existing network models or algorithms that are designed to capture such *structural information* of nodes, which we also categorize into two groups.

Motif-counting structure models. Traditional structure models essentially compares the structures of each pair of networks by counting the numbers of shared motifs in the two networks, which basically characterize the pair-wise similarity in an m-dimensional reproducing kernel Hilbert space. Each dimension of the space is often the (exact or approximated) count of a particular motif. Such a computation, although does not always specify an absolute structure representation, the comparison is inductive and valid across networks from a pure structural perspective. Popular motif-counting structure models include the shortest-path kernel [171], random walk kernel [172], Weisfeiler-Lehman subtree kernel [173] and graphlet kernel [174, 175]. The main difference among these methods lies in the way they define and count the motifs in networks.

Note that, when we use the motif-counting models to compute the *structure* of a node v, they only care about the absolute structures of the surrounding subnetwork $S(v)$. By their definitions, the motif-counting models do not care about the identities of nodes in $S(v)$, which means when the *structures* of two nodes v and u are compared, it does not matter how many (direct or indirect) neighbors v and u share. In other words, the relative *proximity* among v and u does not directly influence their absolute *structures*, and thus the *structure information* is orthogonal with the *proximity information*. However, although characterizing nodes from different perspectives, the two types of information can still correlate, as *positionally close nodes often tend to share similar structures*, while the opposite statement may not be true.

Embedding-based structure models. Intuitively, we conclude that the group of transductive path- and neighborhood-based network embedding models are ineffective in capturing *structure* information, since they only model the overlaps of paths including two nodes

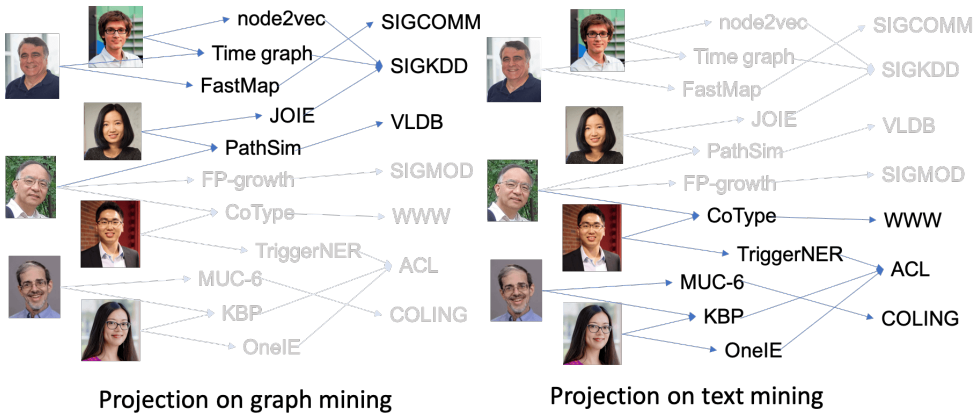


Figure 3.3: Networks projected regarding different venue-field contexts.

v and u or neighborhoods of v and u to compute the relative *proximity* between v and u.

As for GNNs, following our arguments about their universal representation power in the graph spectral domain, they are essentially designed to capture the absolute *structures* of networks. This ability has been increasingly recognized by recent theoretical studies and testified through extensive graph-level classification tasks [168, 169, 170]. To really capture the absolute and inductive *structure* information, it is noted that one should not include any node features that are non-transferrable across compared networks, such as random initializations, one-hot initializations, network specific node features, and transductive proximity embeddings such as spectral embedding and DeepWalk [29]. Instead, pure structural features like node degrees or informationless features like constant vectors are recommended. Also note that, unlike motif-counting models, when using GNNs, ones does not always need to explicitly specify the surrounding subnetwork $S(v)$ of a node v. GNNs can be directly computed on the ambient network G, and the resulting representation of each node can be already regarded as a structural representation of its surrounding neighborhood, with the order of the neighborhood approximately defined by the architecture of the particular GNN model.

Contextualized structure. Under different contexts, the subnetwork structures can persist. Continue with our toy example on DBLP. Figure 3.3 shows the projections of the network in Figure 3.1 under different projections regarding venue-field. The exact networks are different when different contextualized projections are applied (*i.e.*, comprised of different nodes and links). However, some common structures might be persisting and shared across different subnetworks, which motivates proper communications and joint training of different local models.

In Table 3.2, we compute the PathStruct similarity between Jiawei Han and the same other

Context	Jure Leskovec	Christos Faloutsos	Yizhou Sun	Xiang Ren	Heng Ji	Ralph Grishman
venue-field: data	0.9997	0.9999	0.9939	0.9760	0.9466	0.7192
venue-field: ml	0.9982	0.8087	0.8087	0	0.8087	0.8087
venue-field: nlp	0.9939	0.9080	0.9932	0.9080	0.9614	0.9584
venue-field: ir	0.9154	0.9999	0.9933	0.7411	0.9457	0.9154
venue-field: bio	0	0.9806	0.9806	0	0	0
paper-year: <2000	0	0.9986	0	0	0	0.9999
paper-year: 2000-2010	0.9878	0.9996	0.8445	0.7351	0.9814	0.9919
paper-year: >2010	0.9999	0.9999	0.9906	0.9687	0.9965	0.9901
paper-cite: <5	0.9958	0.9991	0.8601	0.9581	0.9625	0.9915
paper-cite: 5-20	0.9279	0.9960	0.8952	0.9442	0.9979	0.9538
paper:cite >20	0.9990	0.9999	0.9905	0.8912	0.9923	0.9984

Table 3.2: PathStruct similarity (A-P-A and A-P-V-P-A) between Jiawei Han and other authors on DBLP under different contextualized projections.

authors in the same real-world DBLP dataset as used in Table 3.1. Note that, the similarity measure is different, as we now care about the structures of authors' ego-networks regarding the meta-paths A-P-A and A-P-V-P-A, *i.e.*, the cosine similarity between the two-dim vectors of numbers of meta-paths directly connected to the authors. Thus the PathStruct similarity is higher if two authors have similar collaboration styles (number of co-authors in each paper) and venue preference (size of venues). We use this metric as a simplification of the various existing kernel based structure measures mentioned before, to simply showcase the possibly different structures of subnetworks under different contexts. Being different from the results in Table 3.1, the results here are again intuitive yet insightful. In particular, most of the scores are high (> 0.9), because the set of authors all have pretty similar collaboration styles and venue preferences. Nonetheless, the similarities between same pairs of authors are slightly different across different contexts. For example, Christos Faloutsos is closest to Jiawei Han under the fields of data mining and IR, and their styles become closer and closer in recent years; Yizhou Sun is closest to Jiawei Han in data mining, and the most popular papers they publish have similar styles (regarding collaboration and venue); Heng Ji and Ralph Grishman are further to Jiawei Han compared with the other authors, and most close to him in NLP, because they are both NLP researchers.

With the examples and analyses above, it is clear that context is important for the modeling of network structures in real-world networks. The key question here is, can we find a mapping between contexts and structures, so as to infer the contexts from structures, and generate structures from contexts? To this end, we study the novel problem of contextualized and controllable network generation, leading to successful models like CondGen (Section 4.4-4.6).

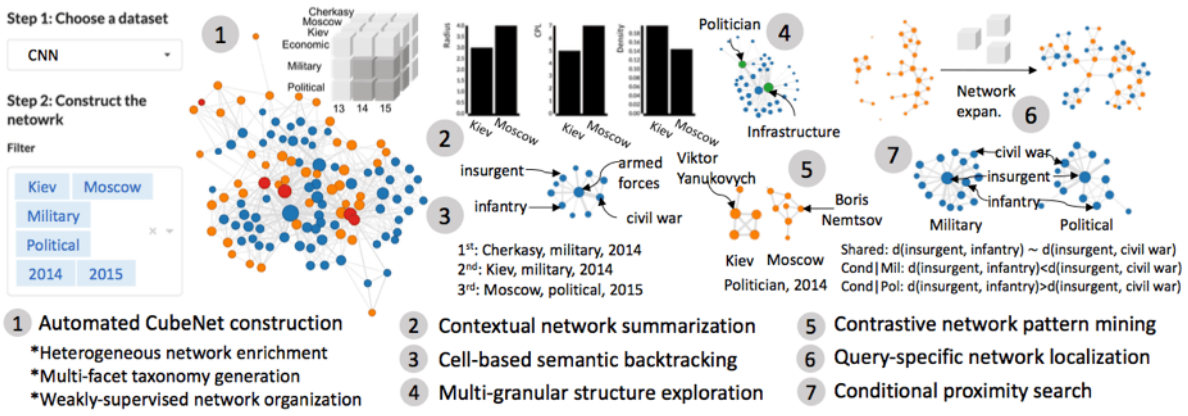


Figure 3.4: The CubeNet system demonstrated at KDD'19 [11].

3.2 A REAL-WORLD SYSTEM

My dissertation research has also led to the implementation of a novel data mining system called Cube Networks (CubeNet in short), which joins the power of data cubes and networks to model the multi-facet contexts and network structures in a principled way.

CubeNet: A principled multi-facet contextualized network mining system. Data cube provides an efficient and systematic way of organizing data *w.r.t.* multi-facet properties and is widely used in traditional data mining over large sets of isolated objects (*e.g.*, records in relational databases, documents in text collections). With context-aware cube structure designs and objects assigned into the multi-facet cells, it largely boosts various downstream data analysis and mining tasks. On the other hand, network serves as a generic and flexible model for object interactions (*e.g.*, document references, gene interactions). By properly integrating the two worlds, I propose the novel data model of CubeNet, which immediately enables various insightful unique functions such as multi-facet multi-granular network exploration (*e.g.*, visualize and compare user connection structures in hierarchical communities formed due to different reasons), contrastive network pattern mining (*e.g.*, finding abnormal gene-protein pathways along the develop of a cancer), cell-based network context backtracking (*e.g.*, given a set of interconnected sensors, retrieve the top-k conditions as context combinations under which the sensors are likely to be correlated in the same way), and so on. A prototype system implementing these functions, as outlined in Figure 3.4, has been demonstrated in KDD'19 [11] and attracted a large audience. Moreover, the design of CubeNet further sets up a novel graph mining paradigm, which poses various new challenges and opportunities around the principled construction, mining and application of CubeNet.

In the example of Figure 3.4, a large multi-facet heterogeneous network is organized *w.r.t.* a topic-location-year data cube structure, which leverages the particular construction steps and enables the unique data analysis and mining functions as follows.

3.2.1 Construction

Heterogeneous network enrichment. Without clear semantics, real-world networks are less informative. For more insightful data analysis and mining, we enrich the heterogeneity of networks by incorporating nodes from massive free texts. In this system, we leverage our recent research on text mining, *i.e.*, AutoPhrase [176] for phrasal node extraction and AutoNER [136] for typed link generation.

Multi-facet taxonomy generation. In this system, we leverage both existing metadata and our recent research on automatic taxonomy generation, *i.e.*, TaxoGen [177], to create multiple taxonomies for each network, essentially leading to a data cube structure [178].

Weakly-supervised network organization. To organize networks into the data cube structure, *i.e.*, allocate nodes to proper cells, we leverage our recent research on heterogeneous network classification based on AutoPath [5], which assigns similar labels in the taxonomy to nodes close in the network based on small sets of nodes weakly labeled via surface name matching.

3.2.2 Analysis

Contrastive network summarization. Various traditional network statistical measures such as clustering coefficient, character path length and triangle count become hard to compute and meaningless in massive universal networks. However, in CubeNet where each cell holds a relatively small network, the structures can be efficiently summarized and contrasted across relevant cells by aggregating network statistics, which provides insight into network evolution along different semantic dimensions (Ex. 2 in Figure 3.4).

Cell-based semantic backtracking. While text cube supports the retrieval of most relevant cells *w.r.t.* unary queries, CubeNet further allows semantic backtracking *w.r.t.* network queries, such as pairs of nodes and small sub-networks. The idea is to combine the graph coverage [179] and top-k cell search [180] algorithms to find the k cells that mostly cover the network query from all cells with an optimized search order (Ex. 3 in Figure 3.4).

Multi-granularity structure exploration. By allocating nodes into hierarchically organized cells, CubeNet supports network roll-up and drill-down, which essentially merges nodes and edges into super-nodes and super-edges (or the other way around), to allow the exploration of network structures in preferred granularities. To make the process efficient, we implement the techniques developed in our previous research on graphcube [181] (Ex. 4 in Figure 3.4).

3.2.3 Mining

Contextual network pattern mining. Traditional graph pattern mining does not consider the contexts of networks. To find more semantically related patterns, we extend [182] to CubeNet by computing a mixture score of popularity, integrity and distinctiveness. Popularity is computed as the normalized frequency, integrity is the ratio of frequency between the pattern and its corresponding close pattern, and distinctiveness is the ratio between the frequency in a cell and the average of all cells. They are combined using customized weights to highlight user preference (Ex. 5 in Figure 3.4).

Query-specific network localization. Given data mining queries over particular sets of nodes, computation over the universal massive network is wasteful and hard to handle. Since CubeNet organizes networks by grouping semantically relevant nodes, it is possible to find a set of cells that mostly cover the queried nodes. To this end, we apply our on-going research on query-specific network construction, which leverages a light reinforcement learning algorithm to find the optimal combination of cells, from which a relevant and complete network can be constructed to support downstream data mining on queried set of nodes (Ex. 6 in Figure 3.4).

Conditional proximity search. Recent research on network embedding often does not easily scale to massive networks and fail to model proximity under different conditions. To deal with both challenges, we apply our on-going research on the co-embedding of network nodes and cube cells, which jointly learns node embedding in each sub-network and a set of embedding transformation functions that align relevant sub-networks. The embedding of sub-networks thus facilitates proximity search conditioned on cell semantics, while the alignment functions enable proximity transfer among similar cells (Ex. 7 in Figure 3.4).

CHAPTER 4: TECHNIQUES

In this Chapter, I present two representative graph mining techniques under the framework of contextualized projections, *i.e.*, graph embedding based on contextualized proximities (Section 4.1-4.3) and graph generation based on contextualized structures (Section 4.4-4.6).

4.1 GRAPH EMBEDDING BASED ON CONTEXTUALIZED PROXIMITIES

Representation learning has become the backbone of various tasks in artificial intelligence [183]. Unsupervised learning is often the default setting due to the desired generalizability. However, many recent works in various fields have demonstrated the profit of leveraging limited label data to learn representations that are not only powerful for the corresponding predictive objectives, but also transferrable to other related tasks [43, 32, 184, 185, 186]. Among them, hierarchical labels residing in given taxonomies have been widely used for natural language processing and bioinformatics, which are especially useful for the tasks of hypernym modeling and hierarchical classification [187, 188, 189, 143, 190, 191, 192]. In their essence, these methods jointly learn the representations of objects and labels in a shared latent space. The objects they model often have rich features, but they do not directly interact with each other.

As for representation learning on networks of interconnected objects (nodes), intensive research has been done on the modeling of both plain networks without node features [29, 26, 31, 41, 193, 194] and content-rich networks with node attributes and/or labels [195, 196, 32, 197, 198]. Recently, the notion of taxonomy has been explored by pioneering research [199, 200], which assume and seek for the latent hierarchical structure underlying the seemingly flatly connected nodes. However, without proper reference to a particular underlying taxonomy, the learned network embedding is still limited to global network mining tasks and uninterpretable without further analysis [201].

Thanks to the vast effort in taxonomy construction from both the research community [177, 202] and industry, increasing amount of network data nowadays can be readily associated with existing taxonomies (Sec. 4.3.1), which provides great opportunities for enhancing network embedding (Sec. 4.3.2) and enabling novel network mining tasks (Sec. 4.3.3). Meanwhile, the rich relational data in networks may also help in better modeling and interpreting the existing taxonomies (Sec. 4.3.4).

Consider a toy example in Figure 4.1, which consists of an author network (*e.g.*, given

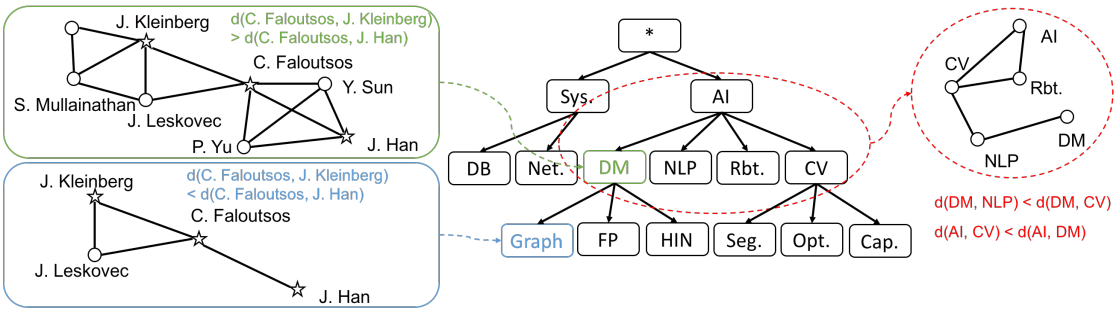


Figure 4.1: Toy example of TaxoGAN: Authors in a publication network are naturally connected to a research topic taxonomy. Through proper modeling of *conditional node proximity* (on the left side) and *hierarchical label proximity* (on the right side), we aim to leverage author node proximity in the network to capture topic label proximity in the taxonomy, which in turn can benefit the learning of both author and topic representations in a closed loop.

by DBLP[1]) and a research topic taxonomy (*e.g.*, given by ACM[2]). Author-author links can be generated *w.r.t.* co-authorships, while author-label links can be generated by keyword matching between the topic names in the taxonomy and the published papers of the authors. In this work, we stress the importance of two novelly observed properties, *i.e.*, *conditional node proximity* and *hierarchical label proximity*.

Conditional node proximity. While existing works on network embedding mostly consider network proximity within the same set of nodes, we argue that node proximity should be conditionally measured within the proper context. For example, on the left side of Figure 4.1, consider the proximity between C. Faloutsos and J. Kleinberg (particularly, in comparison to that between C. Faloutsos and J. Han). When working on Graph Mining (Graph) problems, C. Faloutsos and J. Kleinberg share more important coauthors like J. Leskovec, thus resulting in a smaller distance. However, when working on broader problems in Data Mining (DM), they find their own coauthors like S. Mullainathan and J. Han from different fields, hence resulting in a larger distance. As such, under different conditions, node proximity can be rather different and even contradictory.

As we will show in more details in Sec. 4.2, although a given taxonomy naturally allows for the construction of various conditional subnetworks, the modeling of conditional node proximity is non-trivial. This is because modeling all conditional subnetworks separately will prohibit the leverage of node interactions across different subnetworks and suffer from data sparsity, but modeling all conditional subnetworks together in a flat way will lead to a cluttered embedding space violating the hierarchical label relations.

[1]https://dblp.uni-trier.de/
[2]https://dl.acm.org/ccs/ccs.cfm

Hierarchical label proximity. Although we assume the existence of given taxonomies for particular networks, where node labels are organized in tree-structured hierarchies, the actual distribution and relative distance of these labels in the embedding space is unknown. For example, consider the four labels CV, NLP, Rbt. and DM on the right side of Figure 4.1. Although they are all child labels of the parent label AI, the distances among these siblings as well as their distances to AI might be rather different, which is impossible to understand by solely looking at the taxonomy structure itself. In this work, we propose to leverage the rich relational information from the networks to model the fine-grained proximity among the hierarchical labels. Continue with our example. Since authors working on Rbt. may overlap or collaborate more with those working on CV than DM, the distance between Rbt. and CV should be smaller than that between Rbt. and DM. Moreover, compared with authors working on DM, authors working on CV might more often study the core problems of AI. As a consequence, the distance between AI and CV should be smaller than that between AI and DM.

As we will discuss more in Sec. 4.2, proper modeling of the hierarchical label proximity can further help in regularizing the network embedding of all nodes. However, the task is again non-trivial, as the embedding distances in different hierarchical levels should not be modeled in the same space, but how proximity information can be transferred across the different spaces is unclear.

Present work. We propose TaxoGAN to co-embed network nodes and hierarchical labels, which leverages stacked generative adversarial nets to model the conditional node proximity and hierarchical label proximity in networks associated with label taxonomies. Specifically, TaxoGAN models a hierarchical network generation process, where a network generator is devised at each parent label in the taxonomy to model the children network induced by the corresponding child labels and labeled nodes in the original network. Moreover, a learnable network encoder is devised at each child label to enable the learning of proximity transfer from the embedding spaces of children to parents in a fine-to-abstract manner along the actual label paths in the taxonomy. Finally, we device hierarchical adversarial learning to achieve efficient and robust model inference.

The main contributions of this work are summarized as follows.

1. We propose and formulate the novel problem of co-embedding network nodes and hierarchical labels, where we particularly model the two novel important properties of conditional node proximity and hierarchical label proximity (Sec. 4.2.1).

2. We develop TaxoGAN to model hierarchical network generation under given taxonomy. TaxoGAN simultaneously improves network node embedding and enables hierarchical

label embedding by leveraging induced label networks and proximity transfer in the taxonomy (Sec. 4.2.2-3).

3. We prepare four real-world datasets by linking networks with given taxonomies and conduct thorough experiments regarding the tasks of hierarchical node classification and link prediction. Significant improvements on both tasks compared with popular and state-of-the-art network embedding algorithms demonstrate the power of TaxoGAN on improving the quality of network embedding (Sec. 4.3.1-2).

4. We design two novel tasks of *conditional proximity search* and *taxonomy visualization*, and conduct insightful case studies to demonstrate the unique utility and interpretability of TaxoGAN (Sec. 4.3.3-4).

4.2 MODEL: TAXONOMY-GUIDED GRAPH ADVERSARIAL EMBEDDING (TAXOGAN)

4.2.1 Problem Formulation

Input. We take the input of a network $\mathcal{N} = \{\mathcal{V}, \mathcal{E}, \mathcal{Y}\}$ and a taxonomy $\mathcal{T} = \{\mathcal{L}, \mathcal{H}\}$, where $\mathcal{V} = \{v_i\}_{i=1}^{N}$ is the set of nodes, $\mathcal{E}$ is the set of node-node links, $\mathcal{Y}$ is the set of label-node assignments, $\mathcal{L} = \{l_j\}_{j=1}^{M}$ is the set of labels, and $\mathcal{H}$ is the set of label-label links. For simplicity, we consider uniform undirected node-node links in $\mathcal{E}$, while our model easily generalizes to networks with weighted directed links. By the definition of taxonomy, label-label links in $\mathcal{H}$ are uniform and directed, pointing from parent labels to child labels. Our model works for taxonomies both in tree and DAG structures.

$\mathcal{Y}$ serves as the bridge between $\mathcal{N}$ and $\mathcal{T}$, where for each node $v_i \in \mathcal{V}$, $\mathbf{y}_i$ is the set of labels assigned to v_i. In this work, we require all labels in $\mathbf{y}_i$ to also appear in $\mathcal{L}$, but $\mathbf{y}_i$ can be empty or include any combination of multiple labels. In other words, we only consider node labels organized in a given taxonomy, while we allow the links between the network and the taxonomy to be flexible (likely also weak and noisy). Due to the rapid development of taxonomy construction methods and the growing availability of real-world taxonomies, many networks can be naturally connected with existing taxonomies, which leads to an urge in developing proper models for the joint modeling of both worlds.

Output. To effectively capture the interactions among nodes in the network and labels in the taxonomy, we propose to co-embed $\mathcal{V}$ and $\mathcal{L}$. Therefore, the output of TaxoGAN consists of an $(N \times K)$-dim embedding matrix $\mathbf{U}$ for $\mathcal{V}$ and an $(M \times K)$-dim embedding matrix $\mathbf{Q}$ for

$\mathcal{L}$. As we will show later, although we embed $\mathcal{V}$ and $\mathcal{L}$ as the same dimension, their proximity is preserved in different projected spaces, which is necessary for capturing the conditional node proximity under different contexts. Moreover, the projected spaces are connected via learnable transformation functions, which effectively learns to transfer proximities along parent-child label links and captures the hierarchical label proximity.

4.2.2 Preliminaries

Heterogeneous graph embedding. A naïve way to jointly model $\mathcal{N}$ and $\mathcal{T}$ is to use a heterogeneous graph, where labels are flattened in the taxonomy. PTE [43] provides a vanilla formula to embed such graphs. In our case, consider nodes $\mathcal{V}$ in $\mathcal{N}$ as *words* with undirected co-occurrence links and labels $\mathcal{L}$ in $\mathcal{T}$ as *documents* connected by directed *citation links*. A heterogeneous graph of $\mathcal{V}$ and $\mathcal{L}$ can be embedded according to the following objective

$$J_{PTE} = J_{vv} + J_{vl} + J_{ll}, \tag{4.1}$$

where

$$\begin{aligned} J_{vv} &= -\sum_{e_{ij}\in\mathcal{E}} w_{ij} \log \mathcal{G}(v_i, v_j), \\ \text{with} \quad \mathcal{G}(v_i, v_j) &= \frac{\exp(\mathbf{u}_i'^T \cdot \mathbf{u}_j)}{\sum_{v_k\in\mathcal{V}} \exp(\mathbf{u}_k'^T \cdot \mathbf{u}_j)}; \end{aligned} \tag{4.2}$$

$$\begin{aligned} J_{vl} &= -\sum_{y_{ij}\in\mathcal{Y}} w_{ij} \log \mathcal{G}(v_i, l_j), \\ \text{with} \quad \mathcal{G}(v_i, l_j) &= \frac{\exp(\mathbf{u}_i'^T \cdot \mathbf{q}_j)}{\sum_{v_k\in\mathcal{V}} \exp(\mathbf{u}_k'^T \cdot \mathbf{q}_j)}; \end{aligned} \tag{4.3}$$

$$\begin{aligned} J_{ll} &= -\sum_{h_{ij}\in\mathcal{H}} w_{ij} \log \mathcal{G}(l_i, l_j), \\ \text{with} \quad \mathcal{G}(l_i, l_j) &= \frac{\exp(\mathbf{q}_i'^T \cdot \mathbf{q}_j)}{\sum_{l_k\in\mathcal{L}} \exp(\mathbf{q}_k'^T \cdot \mathbf{q}_j)}. \end{aligned} \tag{4.4}$$

Each $\mathcal{G}(o_i, o_j)$ models the probability of generating a linked from object (node/label) o_i to object o_j. Following the setting of Skip-gram adapted to network embedding [29, 26], we use $\mathbf{U}/\mathbf{Q}$ as the target embedding and $\mathbf{U}'/\mathbf{Q}'$ as the context embedding, which allows explicit modeling of the second-order proximity as proposed in LINE [26].

To optimize Eq. 4.1, stochastic gradient descent with the techniques of edge sampling and negative sampling [26] can be leveraged. However, the random negative sampling process

does not leverage the graph structures at all, which leads to inefficient and unstable training.

Adversarial graph embedding. To enable efficient and robust graph embedding, GraphGAN [31] was proposed based on the concept of adversarial learning. Instead of randomly sampling negative pairs of objects (objects without links), GraphGAN constructs a link discriminator $\mathcal{D}$ and a fake link generator $\mathcal{G}$, and iteratively optimizes the following objective function by allowing $\mathcal{D}$ and $\mathcal{G}$ to play a two-player *minimax* game

$$\min_{\theta_G}\max_{\theta_D} J_{gGAN} = \sum_{v_i\in\mathcal{V}}\Big(\mathbb{E}_{v\sim p_{true}(\cdot,v_i)}\big[\log\mathcal{D}(v,v_i;\theta_D)\big] + \mathbb{E}_{v\sim\mathcal{G}(\cdot,v_i;\theta_G)}\big[\log\big(1-\mathcal{D}(v,v_i;\theta_D)\big)\big]\Big). \tag{4.5}$$

Empowered by the novel graph softmax function, $\mathcal{G}$ is able to efficiently generate strong negative samples on-the-fly during training in a graph-structure-aware way [31]. Note that, by sharing the target and context embedding in both $\mathcal{G}$ and $\mathcal{D}$, GraphGAN does not explicitly consider second-order proximity as in PTE and LINE [26]. However, since $\mathcal{G}$ and $\mathcal{D}$ still maintain two sets of embedding, which takes charge of generating and discriminating links respectively, GraphGAN manages to outperform LINE on classic network embedding tasks by significant margins.

In another line of research, complex generative adversarial nets (GAN) have been rapidly developed in domains like computer vision and natural language processing. Particularly, we notice the SGAN model developed for hierarchical image representation learning [203], which consists of a top-down stack of GANs learned to generate high-level to low-level image representations in a hierarchical fashion. While the tasks of image representation and network representation are naturally different, we find essential connections between their task and ours, due to the consideration of underlying hierarchical structures.

4.2.3 TaxoGAN

In this work, we aim at co-embedding network nodes and hierarchical labels. To understand the main challenges of this task, let us take a look at Figure 4.2, where an author node J. Leskovec has three labels Graph, DM and AI in the research topic taxonomy. In this simple case, on one hand, if we do not consider the hierarchical structure of labels and put them all in a *single* space, the author embedding will eventually lie somewhere in the middle of the three label embeddings (as marked by the blue '+' sign), which violates the label hierarchy and results in underfitting. On the other hand, if we simply use *separate* spaces to embed the nodes and labels under each parent label, the model will ignore the rich correlations among

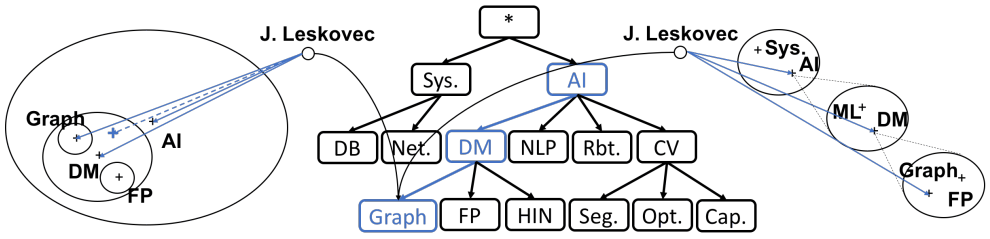


Figure 4.2: Illustration of the main challenges: Modeling network nodes and hierarchical labels all together in a single space leads to a cluttered embedding space violating the underlying hierarchy, while simply partitioning them into separate spaces ignores label correlations and results in parameter redundancy.

labels in the hierarchy, bringing in massive redundant parameters and leading to overfitting.

To address the above challenges, we propose TaxoGAN to co-embed network nodes and hierarchical labels through a hierarchical network generation process, where a network generator is devised at each parent label in the taxonomy to model the subnetwork of nodes and child labels, and a network encoder is devised at each child label to learn the transferrable proximity across levels in the taxonomy. The generator and encoder are jointly trained through efficient and robust hierarchical adversarial learning, where a network discriminator is devised in each embedding space to enforce correct node-node and node-label proximity. In the following, we motivate and describe each component of TaxoGAN in details.

Label-wise subnetwork generator: jointly model node and label proximities in conditional subnetworks. To properly model conditional node proximity and respect the label hierarchy, we propose to generate a specific node-label network under each parent (non-leaf) label in the taxonomy. Let l_p denote an arbitrary parent label in $\mathcal{T}$, and $\mathcal{L}_p$ denote the set of all immediate child labels of l_p. Then $\mathcal{V}_p$ is the subset of $\mathcal{V}$ consisting of all nodes with label l_p or labels in $\mathcal{L}_p$. A *conditional subnetwork* $\mathcal{B}_p$ is constructed from $\mathcal{V}_p$, $\mathcal{L}_p$ as well as the node-node links $\mathcal{E}_p$ among nodes $\mathcal{V}_p$ and node-label links $\mathcal{Y}_p$ between nodes $\mathcal{V}_p$ and labels $\mathcal{L}_p$.

$\mathcal{B}_p$ acts as a bridge between node proximity and label proximity under the condition of l_p. In the corresponding embedding space $\mathcal{S}_p$, $\mathcal{V}_p$ and $\mathcal{L}_p$ can then be arranged in a flat way. To learn the node embedding $\mathbf{U}^p$ and label embedding $\mathbf{Q}^p$ in the space of $\mathcal{S}_p$, we devise a subnetwork generator $\mathcal{G}$ to enforce $\mathcal{E}_p$ and $\mathcal{Y}_p$ based on the softmax function as follows

$$\mathcal{G}(v_j, v_i|l_p) = \frac{\exp(\mathbf{u}_j^{pT} \cdot \mathbf{u}_i^p)}{\sum_{v_k \in \mathcal{V}_p} \exp(\mathbf{u}_k^{pT} \cdot \mathbf{u}_i^p)}, \tag{4.6}$$

$$\mathcal{G}(l_s, v_i|l_p) = \frac{\exp(\mathbf{q}_s^{pT} \cdot \mathbf{u}_i^p)}{\sum_{l_k \in \mathcal{L}_p} \exp(\mathbf{q}_k^{pT} \cdot \mathbf{u}_i^p)}. \tag{4.7}$$

Following LINE [26], we can use negative sampling to compute the softmax in Eq. 4.6, since the number of nodes $|\mathcal{V}_p|$ can be quite large even in the subnetworks. However, since the number of child labels $|\mathcal{L}_p|$ is often quite small, we can directly compute the softmax in Eq. 4.7 for better label accuracy. Note that, in each conditional subnetwork, there exist no direct links among labels. Thus, the fine-grained relative distances among child labels under each parent label are learned based on the corresponding network structure, which cannot be inferred from the taxonomy structure itself.

Cross-level learnable encoder: proximity transfer and parameter sharing in the taxonomy. The generator $\mathcal{G}$, without the consideration of label correlations and transferrable information in the taxonomy, can either model all conditional subnetworks essentially in a single embedding space or separately in independent spaces. The key difference lies in the computation of $\mathbf{U}^p$ and $\mathbf{Q}^p$. Since in each conditional subnetwork $\mathcal{B}_p$, we co-embed nodes $\mathcal{V}_p$ and labels $\mathcal{L}_p$ in the space $\mathcal{S}_p$, $\mathbf{U}^p$ and $\mathbf{Q}^p$ can be computed from $\mathbf{U}$ and $\mathbf{Q}$ in the same way. Without loss of generality, we will focus our discussion on the computation of $\mathbf{U}^p$.

Particularly, if $\mathbf{U}^p = \mathbf{U}$, which is shared across all conditional subnetworks, all nodes and labels are essentially flatly arranged in a single embedding space of $\mathbf{U}$, which violates the label hierarchy, resulting in clutter embedding space and underfitting. Otherwise, if we compute a completely different $\mathbf{U}^p$ for each conditional subnetwork, the subnetworks are modeled in independent spaces, which ignores label correlations, leading to large parameter redundancy and overfitting.

As a remedy to this trap, we propose to compute each $\mathbf{U}^p$ as an *encoded version* of $\mathbf{U}$, *i.e.*, $\mathbf{U}^p = \mathcal{A}(\mathbf{U}, l_p)$, so as to essentially transfer proximities captured by different subnetwork generators in the taxonomy. However, since the semantic information in taxonomies is coarse, it is hard to decide how to exactly transfer the proximities. For example, consider the sibling labels of NLP and CV under parent AI. Since NLP communities might be *tighter* than CV as including less diverse subtopics, it should transfer stronger proximity signals. That is, in the subspace of AI, authors close in the subspace of NLP should be closer than those close in the subspace of CV. To capture such subtle semantics in the taxonomy, we require the encoder $\mathcal{A}$ to be *learnable* and *label-dependent*. To this end, we leverage the simple but powerful

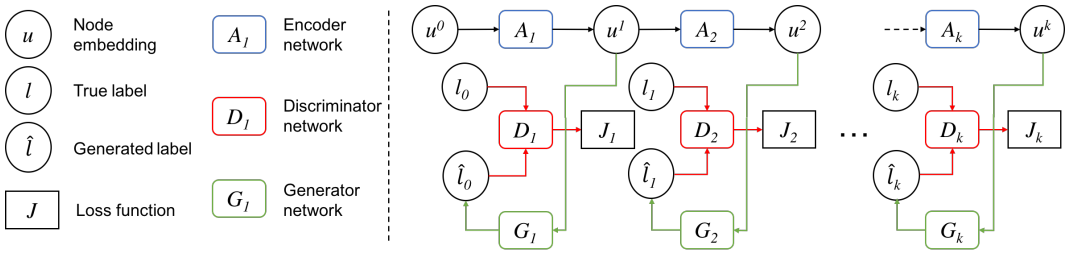


Figure 4.3: TaxoGAN overview: A framework for the adversarial learning of hierarchical network embedding.

nonlinear fully connected feedforward neural network (FNN) to model $\mathbf{U}^p$ as

$$\mathbf{U}^p = \mathcal{A}(\mathbf{U}, l_p) = \text{ReLU}(\mathbf{A}_p\mathbf{U}) + \mathbf{b}_p, \tag{4.8}$$

where $\mathbf{A}_p$ and $\mathbf{b}_p$ are the learnable parameters in the encoder at l_p.

Learning a separate encoder function at each child label does not really leverage the hierarchical structure of $\mathcal{T}$ and still leads to large parameter spaces. To this end, we get motivated by the idea of hierarchical image representation learning [203], which leverages stacked encoders to guide the generation of image representations from high (abstract) to low (detailed) levels. In our scenario, since nodes in the network are connected with labels in the taxonomy, they can also be described by representations at multiple granularities [200]. Therefore, we propose to parameterize $\mathcal{A}$ as *nested embedding transformations* following the hierarchy paths along the taxonomy. For any label l_p, let $l_p \to \ldots \to l_j \to l_i$ denote the path from l_p to a certain leaf label l_i. We have

$$\mathbf{U}^p = \mathcal{A}(\mathbf{U}, l_p) = \mathcal{A}_p(\cdots \mathcal{A}_j(\mathbf{U}, l_j) \cdots, l_p). \tag{4.9}$$

Note that, the number of parameters in $\mathcal{A}$ grows linearly with the number of labels $|\mathcal{L}|$ in the taxonomy. However, since the main purpose for using $\mathcal{A}$ is to compute multi-granularity node embeddings and separate labels on different levels, it is reasonable to share the parameters of $\mathcal{A}$ among all labels on the same levels of the taxonomy, which reduces the model complexity of $\mathcal{A}$ to $\log |\mathcal{L}|$, and further alleviates possible overfitting due to sparse data in certain subspaces.

Adversarial network discriminator: enable efficient and robust learning. Through subnetwork generation and learnable encoding, we essentially manage to partition the whole network and taxonomy into a set of conditional subnetworks with proper proximity transfer functions. Following the classic heterogeneous network embedding framework of Eq. 4.1, we

formulate the overall objective of TaxoGAN into

$$J_{\text{TaxoGAN}} = J_{vl} + \lambda_1 J_{vv} + \lambda_2 J_{ll}, \tag{4.10}$$

where each of J_{vv}, J_{vl} and J_{ll} is only slightly different from those in Eq. 4.1 by replacing the global generator $\mathcal{G}$ with conditional generators and embedding encoders defined in Eq. 4.6-4.9.

In practice, we find the joint training of generator networks $\mathcal{G}$ and encoder networks $\mathcal{A}$ to be often inefficient and unstable. Inspired by recent advances in adversarial learning [31, 41, 193, 194], we propose to improve the efficiency and robustness of model inference, by designing a novel hierarchical adversarial network discriminator $\mathcal{D}$. Specifically, each of J_{vv}, J_{vl} and J_{ll} can be optimized through a two-player minimax game defined in Eq. 4.5, with the corresponding designs of $\mathcal{G}$ and $\mathcal{A}$ defined in Eq. 4.6-4.9 and $\mathcal{D}$ defined as follows, which measure the log-probability of node-node and node-label links.

$$\mathcal{D}(v_j, v_i|l_p) = \frac{1}{1 + \exp(-\mathbf{u}_j^{pT}\mathbf{u}_i^p)}, \tag{4.11}$$

$$\mathcal{D}(l_s, v_i|l_p) = \frac{1}{1 + \exp(-\mathbf{q}_s^{pT}\mathbf{u}_i^p)}. \tag{4.12}$$

As illustrated in Figure 4.3, for each node v_i in $\mathcal{N}$, we consider a bottom-up node encoding process together with a top-down network generation process. $\mathbf{u}_0 = \mathbf{u}$ is the lowest level node embedding, capturing raw node proximity in $\mathcal{N}$. At each parent label l_p in $\mathcal{T}$, the encoder network $\mathcal{A}$ computes a transformed embedding $\mathbf{u}^p$, which ideally can best characterize the embedding of $\mathcal{V}_p$ and $\mathcal{L}_p$ in the conditional embedding space $\mathcal{S}_p$. To achieve this goal, the generator network $\mathcal{G}$ takes $\mathbf{u}_p$ as input and generates the most misleading linked node $\hat{v}_j$ from $\mathcal{V}_p$ and the most relevant label $\hat{l}_s$ from $\mathcal{L}_p$ based on Eq. 4.6 and 4.7. The discriminator network $\mathcal{D}$ then tries to differentiate $\hat{v}_j$ and $\hat{l}_s$ from the true linked nodes and labels by maximizing Eq. 4.10 *w.r.t.* the above equations.

Note that, for stable model training, we find it important for $\mathcal{G}$ and $\mathcal{D}$ to maintain two different sets of node and label embeddings, *i.e.*, $\mathbf{U}'$, $\mathbf{Q}'$ for $\mathcal{G}$ and $\mathbf{U}$, $\mathbf{Q}$ for $\mathcal{D}$, which correspond to the *context embedding* and *target embedding* in [29, 26], respectively. Also note that, since we partition the whole network into series of subnetworks, some node-node links across different subnetworks cannot be directly modeled, but they nonetheless carry important proximity information. To deal with this, we add a global node-node proximity module on the base embedding of nodes $\mathcal{U}$, which is implemented by exactly following [31].

Training algorithm. Finally, with the neural architectures of generator $\mathcal{G}$, discriminator

Algorithm 4.1: TaxoGAN Training

Input : network $\mathcal{N}$, taxonomy $\mathcal{T}$, embedding dimension τ, #batches b_{vl}, b_{vv}, b_{ll}, batch size s, negative sampling rate n

1 **while** *not converge* **do**
2 Sample a parent label l_p and construct the subnetwork $\mathcal{B}_p$ **for** $t \leftarrow 1$ *to* b_{vv} **do**
3 Update $\mathbf{U}'^p$ and $\mathbf{U}^p$ by training $\mathcal{G}_{vv}$, $\mathcal{D}_{vv}$, $\mathcal{A}$
4 **for** $t \leftarrow 1$ *to* b_{ll} **do**
5 Update $\mathbf{Q}'^p$ and $\mathbf{Q}^p$ by training $\mathcal{G}_{ll}$, $\mathcal{D}_{ll}$, $\mathcal{A}$
6 **for** $t \leftarrow 1$ *to* b_{vl} **do**
7 Update $\mathbf{U}'^p$, $\mathbf{U}^p$, $\mathbf{Q}'^p$, $\mathbf{Q}^p$ by training $\mathcal{G}_{vl}$, $\mathcal{D}_{vl}$, $\mathcal{A}$
8 **return** $\mathbf{U}$, $\mathbf{Q}$ and $\mathcal{A}$

$\mathcal{D}$ and encoder $\mathcal{A}$ defined, we describe the detailed joint training process of TaxoGAN in Algorithm 1.

In algorithm 4.1, in Line 3, the design and training of $\mathcal{G}_{vv}$ and $\mathcal{D}_{vv}$ in each subnetwork is the same as in the plain networks of [31]; in Line 5, the training of $\mathcal{G}_{ll}$ and $\mathcal{D}_{ll}$ are very similar to those of $\mathcal{G}_{vv}$ and $\mathcal{D}_{vv}$, only by substituting $\mathbf{U}$ with $\mathbf{Q}$, and $\mathcal{A}$ is shared for $\mathbf{U}/\mathbf{U}'$ and $\mathbf{Q}/\mathbf{Q}'$. Since the sampling of v and l is discrete, all generator networks are trained by policy gradient [204]. For example, the gradient of J_{vl} conditioned on l_p *w.r.t.* $\mathcal{G}$ is computed as

$$
\begin{aligned}
&\nabla_{\mathbf{U}',\mathbf{Q}'} J_{vl}|l_p \\
=&\nabla_{\mathbf{U}',\mathbf{Q}'} \sum_{l_j \in \mathcal{L}_p} \mathbb{E}_{l_j \sim \mathcal{G}(\cdot,v_i|l_p)}[\log(1-\mathcal{D}(l_j,v_i|l_p)] \\
=&\sum_{l_j \in \mathcal{L}_p} \mathbb{E}_{l_j \sim G(\cdot,v_i|l_p)} \\
&[\nabla_{\mathbf{U}',\mathbf{Q}'} \log \mathcal{G}(l_j,v_i|l_p) \log(1-\mathcal{D}(l_j,v_i|l_p))].
\end{aligned}
$$

Training the generator networks $\mathcal{G}$ results in the update of $\mathbf{U}'$ and $\mathbf{Q}'$, while training the discriminator networks $\mathcal{D}$ results in the update of $\mathbf{U}$ and $\mathbf{Q}$. For stability concern, we fix $\mathcal{A}$ during the training of $\mathcal{G}$, and only update it while training $\mathcal{D}$.

In each iteration, the complexity of Line 2-3 is $O(b_{vv}snd_NK)$, Line 4-5 is $O(b_{ll}snd_TK)$, Line 6-7 is $O(b_{vl}snd_TK)$, where d_N is the average node degree in $\mathcal{N}$ and d_T is the average number of child labels of each non-leaf label in $\mathcal{T}$. b_{vv}, b_{ll} and b_{vl} are set to balance the trade-off among the three objectives and reflect the weighing parameters λ_1 and λ_2 in Eq. 4.10. Considering convergence to be reached after a constant number of iterations over all nodes,

the overall complexity of TaxoGAN is bounded by the $N \log N$ complexity of global network embedding same as [31].

We implement TaxoGAN with Pytorch. As we can observe from the experimental results, the variance across different trains of TaxoGAN on the same data is not large. We further inspect the loss curves and conclude that the training process of TaxoGAN is stable. All code and data will be released upon the acceptance of this work.

4.3 EXPERIMENTAL EVALUATIONS

4.3.1 Experimental Settings

Datasets. We construct four datasets of real-world networks with explicit taxonomies.

- **DBLP:** We collect the author network[3] with the research topic taxonomy[4]. Undirected uniform links in the network are generated based on coauthorships. A label in the taxonomy is assigned to an author if her/his papers mentions the keyword.
- **Yelp:** We collect the business network[5] with the category taxonomy[6]. Undirected uniform links in the network are generated based on common customers who posted reviews for both businesses. Label assignments are given in the original dataset.
- **FreeBase:** We collect the entity network[7] with the type taxonomy[8]. Undirected uniform links in the network are generated if two entities appear together in any triplet of facts. Labels are assigned by retrieving the nested entity types.
- **PubMed:** We collect the protein network[9] with the disease taxonomy[10]. Undirected uniform links in the network are generated if mentions of two proteins appear in any same sentence. Labels are assigned by surface name matching.

Compared algorithms. We compare with three groups of network embedding algorithms from the state-of-the-art to comprehensively evaluate the performance of TaxoGAN.

[3]https://dblp.uni-trier.de/xml/
[4]https://dl.acm.org/ccs/ccs_flat.cfm
[5]https://www.yelp.com/dataset
[6]https://www.yelp.com/developers/documentation/v3/all_category_list
[7]http://freebase-easy.cs.uni-freiburg.de/dump/
[8]http://dbpedia.org/page/Taxonomy
[9]ftp://ftp.ncbi.nih.gov/pub/taxonomy
[10]ftp://ftp.ncbi.nlm.nih.gov/

Datasets	Network		Taxonomy	
	#nodes	#links	#labels	#levels
DBLP	81,389	208,711	268	4
Yelp	14,573	55,243	438	4
FreeBase	30,180	53,632	18	3
PubMed	9,619	25,655	87	4

Table 4.1: Statistics of the four real-world datasets we use.

- *Plain network embedding:* We compare with DeepWalk [29] and GraphGAN [31]. DeepWalk is the most pioneering and popular Skip-gram based network embedding algorithm, while GraphGAN represents the state-of-the-art plain network embedding models leveraging adversarial learning. We run both algorithms on the original networks by ignoring the taxonomies.

- *Attributed and labeled network embedding:* We compare with PTE [43] and GraphSage [33]. PTE is an extension of the popular LINE [26] algorithm to networks with attributes and labels. We treat taxonomies as flat label networks, and run PTE on the bipartite networks of nodes and labels. GraphSage represents the state-of-the-art attributed and labeled network embedding models. We regard all labels as flat node attributes and train GraphSage in the link prediction fashion.

- *Taxonomy aware network embedding:* We compare with Poincare [199] and Nethiex [200], which are the most recent network embedding algorithms assuming latent node taxonomies. Since they do not work with explicit taxonomies, we run both of them on the original networks as in their original settings.

We also conduct comprehensive ablation study by comparing four different TaxoGAN variants: (1) TaxoGAN-sin is the model with a single embedding space; (2) TaxoGAN-sep is the model with separate embedding spaces; (3) TaxoGAN-noadv is the model without adversarial training; (4) TaxoGAN is our full model.

Evaluation protocols. We evaluate all algorithms on two fundamental tasks: node classification and link prediction.

For node classification, since we consider hierarchical labels in taxonomies in this work, we focus on the setting of level-by-level classification. Given the learned embedding of training nodes and the label taxonomy, we further train a linear SVM at each parent label to classify the testing node *w.r.t.* the current child labels. During testing, each node thus can be assigned to a path in the label taxonomy, a testing node-label pair (v, l) is correct

if the predicted label path of v includes l. All TaxoGAN models except for TaxoGAN-sin use the corresponding embeddings in each level, while the other models all use a single embedding across all levels. We randomly split the set of labeled nodes into training and testing sets with the ratio of 4:1 for five times and compute the testing F1 of each node-label pair. We aggregate the pair-wise F1 scores by each node to compute the micro F1 and by each label to compute the macro F1.

We consider standard link prediction in the same way as in [29, 26]. Predicted links are ranked by the cosine distance among the node embedding vectors. All TaxoGAN models use the shared base embedding $\mathcal{U}$ for link prediction. We randomly split the set of all links in the network into training and testing sets with the ratio of 4:1 for five times and compute the standard AUC and MRR scores over all links in the testing sets.

Parameter settings. The implementations of all compared algorithms are provided by their original authors, and all model hyper-parameters are tuned to the best via standard five-fold cross validation. For TaxoGAN, we use the same parameters for all datasets. Without much tuning, we empirically set the loss weighing parameters λ_1 and λ_2 to 0.1, embedding dimension τ to 50, batch size s to 64 and learning rate to 10^{-4}. All batch numbers b's are set to 128 and negative sampling rate n is set to 5.

4.3.2 Quantitative Evaluations

Table 4.2 presents the performance of compared algorithms on hierarchical node classification. The improvements of TaxoGAN over the second runners all passed the significance t-test with p-value 0.01. Since the classification at each level in the label taxonomy is multi-class, and deeper labels are harder to be correctly predicted (if any precedent label is predicted wrong, the label path can never reach the correct label), the absolute F1 values are all pretty low. Dataset like Yelp has a lot of deep but narrow labels, which are hard to correctly predict, and the mistakes largely impact the macro F1, whereas dataset like PubMed has a lot of shallow but wide labels, and the mistakes largely impact the micro F1. Thus the suite of datasets and metrics provides a comprehensive evaluation towards the compared algorithms.

The baselines have varying performance across different datasets, while PTE and GraphSage often perform better due to the leverage of labeled data during training. By considering latent hierarchies, Poincare and Nethiex perform better than DeepWalk and GraphGAN in many cases, but their learned latent hierarchies do not always perfectly match the reality and even lead to worse performance in some cases like on DBLP.

DeepWalk	11.07 ± 0.61	26.24 ± 0.84	26.41 ± 1.12	10.94 ± 1.06
GraphGAN	16.10 ± 0.55	26.40 ± 1.21	25.97 ± 0.85	13.68 ± 1.28
PTE	16.42 ± 0.47	33.73 ± 0.93	50.27 ± 1.40	12.71 ± 1.64
GraphSage	18.72 ± 1.18	29.06 ± 0.29	45.77 ± 0.60	12.05 ± 1.17
Poincare	13.87 ± 0.51	29.02 ± 1.12	30.43 ± 1.29	12.73 ± 1.90
Nethiex	10.06 ± 0.56	19.44 ± 1.53	35.39 ± 1.37	12.22 ± 1.31
TaxoGAN-sin	20.56 ± 0.25	34.88 ± 0.42	65.36 ± 0.59	11.81 ± 1.13
TaxoGAN-sep	25.80 ± 1.01	28.47 ± 1.04	63.46 ± 0.46	11.98 ± 0.42
TaxoGAN-noadv	29.52 ± 0.79	39.83 ± 1.09	65.79 ± 1.07	16.31 ± 0.22
TaxoGAN	**31.97 ± 1.44**	**41.37 ± 0.58**	**65.98 ± 0.98**	**20.11 ± 1.41**

Algorithm	Link prediction AUC			
	DBLP	Yelp	FreeBase	PubMed
DeepWalk	83.40 ± 0.26	87.93 ± 0.43	64.93 ± 0.35	69.15 ± 1.18
GraphGAN	83.76 ± 0.09	88.51 ± 0.28	65.00 ± 0.67	68.19 ± 1.31
PTE	75.47 ± 0.15	89.10 ± 0.26	63.16 ± 0.52	71.46 ± 0.86
GraphSage	82.63 ± 0.22	85.33 ± 0.56	66.53 ± 0.51	68.20 ± 1.21
Poincare	84.06 ± 0.15	91.60 ± 0.16	68.86 ± 0.35	71.68 ± 0.80
Nethiex	84.41 ± 0.07	92.70 ± 0.26	69.75 ± 0.68	**71.78 ± 0.28**
TaxoGAN-sin	84.14 ± 0.06	92.31 ± 0.31	67.14 ± 0.41	68.00 ± 0.74
TaxoGAN-sep	84.17 ± 0.14	87.47 ± 0.34	63.29 ± 0.65	68.60 ± 0.64
TaxoGAN-noadv	84.56 ± 0.15	92.22 ± 0.39	66.73 ± 0.66	68.95 ± 0.33
TaxoGAN	**85.02 ± 0.25**	**92.92 ± 0.44**	**70.48 ± 0.32**	70.02 ± 1.03

Table 4.2: Performance of all compared algorithms.

Overall, TaxoGAN constantly outperforms all compared algorithms in all cases, with significant margins over the best baseline ranging from 11% to 70%, and the scores all passed t-test with p-value 0.05, demonstrating its superior effectiveness and generalizability. In particular, the improvements of TaxoGAN are more significant when the numbers of labels are larger and the hierarchies of labels are deeper, like with DBLP and Yelp, which supports the appropriate design of our model to leverage the explicit hierarchical structure of associative labels. Note that, while the unsupervised baselines (DeepWalk, GraphGAN, Poincare and Nethiex) do not have access to the node labels in the taxonomy, PTE and GraphSage use the exact same labels as TaxoGAN. This shows TaxoGAN to be effective in modeling hierarchical label spaces, as we will further demonstrate in the ablation study.

For ablation study, our TaxoGAN-sin model has close performance towards the best baselines like PTE, because they are indeed similar only by the difference in adversarial training; our TaxoGAN-sep model does not always outperform TaxoGAN-sin, indicating that even if the evaluation protocol of level-by-level classification may favor multiple embeddings, simply using separate embeddings is not good enough and can harm the performance

due to problems like subnetwork sparsity and overfitting, and TaxoGAN-sep is extremely hard to train due to redundant parameters and large memory cost; our TaxoGAN-noadv model is the nested space model without adversarial training, which outperforms TaxoGAN-sep with significant margins, corroborating the effectiveness of our model design with connected subspaces through base and transformed embeddings; our TaxoGAN model further outperforms TaxoGAN-noadv, directly showing the advantage of our novel hierarchical adversarial training technique.

Table **??** presents the performance of compared algorithms on standard link prediction. Note that, the main goal of TaxoGAN is hierarchical node classification by design, where we leverage network structures to compute the node embeddings as inputs of the hierarchical classifiers. As a result, the base embeddings that we use for the link prediction experiments are mostly decided by the plain network structures and only get slightly influenced during the training of the hierarchical GAN model. Nonetheless, such fine tuning *w.r.t.* hierarchically structured labels is shown to be useful for global (unconditional) link prediction, which leads to very competitive performance compared to the strongest baselines, further corroborating the general utility of TaxoGAN. It is reasonable to expect TaxoGAN to further excel on datasets where links are also generated under different conditions.

We measure the runtimes of all compared algorithms on a server with one GeForce GTX TITAN X GPU and two Intel Xeon E5-2650V3 10-core 2.3GHz CPUs. We observe the runtimes of TaxoGAN to be similar to GraphGAN and GraphSage, while slightly larger than other baselines like PTE and DeepWalk.

4.3.3 Conditional Proximity Search

To illustrate how TaxoGAN is able to capture both global and conditional proximity on networks with hierarchical labels, we select four of the many well-known researchers from different fields related to data mining and extract their hierarchical embeddings computed by TaxoGAN on DBLP.

In Table 4.3, for each author pair, we present their predicted proximity based on the cosine similarity between their global base embeddings as well as the topic-wise conditional embeddings of the top five research topics where the pair is embedded most closely. As we can observe, (1) the global proximity computed by TaxoGAN reflect the reality, where authors working on more similar topics in general are embedded closer. Meanwhile, (2) the conditional proximities are even more accurate and telling, since they provide essential insights into which particular research topics a given pair of authors are likely to collaborate on. Such knowledge, while directly facilitating the unique application of conditional proximity

search as we advocate in this work, is hard to gather without proper joint modeling of the network and associated taxonomy.

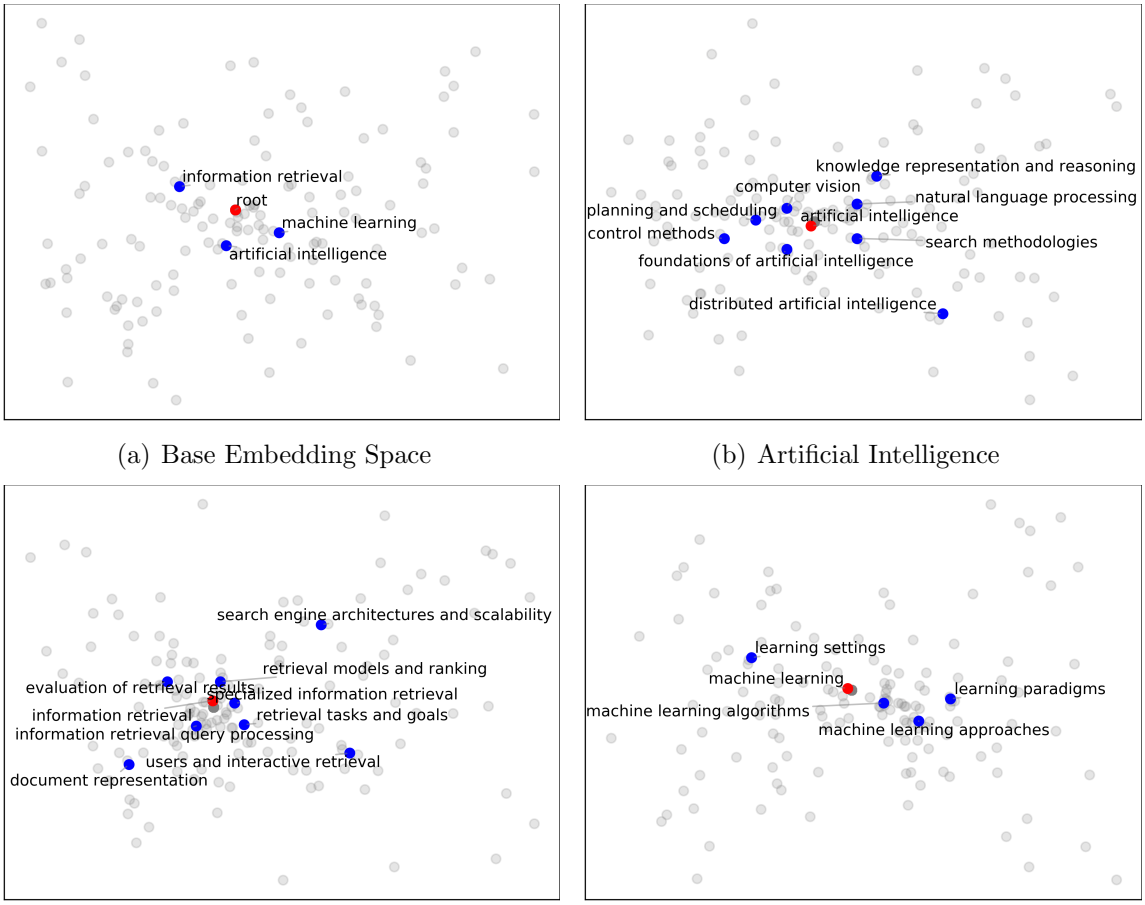
Jiawei Han & Christos Faloutsos	Jiawei Han & Jure Leskovec
global (0.8772)	global (0.7401)
knowledge rep. and reasoning (0.8983)	*information system applications* (0.8238)
collaborative and social computing (0.8884)	*machine learning approaches* (0.7671)
data mining (0.8579)	*collaborative and social computing* (0.7019)
information system applications (0.7960)	*users and interactive retrieval* (0.6784)
spatial-temporal systems (0.7280)	*knowledge rep. and reasoning* (0.5381)
Jiawei Han & Yoshua Bengio	Christos Faloutsos & Jure Leskovec
global (0.6558)	global (0.8883)
foundations of AI (0.7266)	*collaborative and social computing* (0.9229)
retrieval tasks and goals (0.6369)	*specialized information retrieval* (0.8864)
machine learning approaches (0.5616)	*information system applications* (0.8664)
document representation (0.5331)	*search methodologies* (0.8624)
scheduling and planning (0.5162)	*machine learning* (0.7989)
Christos Faloutsos & Yoshua Bengio	Jure Leskovec & Yoshua Bengio
global (0.7939)	global (0.7710)
foundations of AI (0.8143)	*enterprise information systems* (0.7976)
enterprise information systems (0.7319)	*knowledge rep. and reasoning* (0.7313)
collaborative and social computing (0.6996)	*machine learning approaches* (0.7195)
retrieval models and ranking (0.6292)	*planning and scheduling* (0.7044)
computer vision (0.6098)	*search methodologies* (0.6419)

Table 4.3: Pair-wise global and conditional similarity among four researchers jointly learned by TaxoGAN.

4.3.4 Fine-Grained Taxonomy Visualization

Another novel application of TaxoGAN is fine-grained taxonomy visualization, which is enabled by our unique leverage of node proximity in networks associated with the taxonomies. As an example, we visualize the embedding spaces (Figure 4.4, reduced to 2-dim by standard PCA) of four label-induced subnetworks from DBLP, corresponding to the labels root, AI, IR, and ML. Grey dots are nodes in the conditional subnetworks, while red and blue dots are the parent and child labels, respectively. Since many labels have quite similar textual names, such fine-grained label representations are hard to generate by existing methods like word embedding.

As we can observe, the results are highly interpretable and insightful, which provide knowledge about the relative distances among labels. For example, in the AI subnetwork, labels closest to AI include CV, NLP and foundations of AI, while the closest pairs of labels include knowledge rep.–NLP, CV–planning, planning–control, *etc.*. While existing taxonomies mostly



(a) Base Embedding Space

(b) Artificial Intelligence

(c) Information Retrieval

(d) Machine Learning

Figure 4.4: Visualization of the hierarchical label spaces learned by TaxoGAN given the network and label taxonomy.

only include a label skeleton, such label embeddings are valuable towards the understanding of subtle label relations, and likely useful for more downstream tasks like taxonomy refinement and others involving machine learning on taxonomies.

4.4 GRAPH GENERATION BASED ON CONTEXTUALIZED STRUCTURES

Graphs (networks) provide a generic way to model real-world relational data, such as entities in knowledge graphs, users in social networks, genes in regulatory networks, *etc.*. It is thus critical to study the generation of graph structures, which is fundamental for the understanding of their underlying functional components and creation of meaningful structures with desired properties. Nowadays, contextual data like attributes and labels are becoming ubiquitous in networks [11], the rich semantics of which may well correspond to

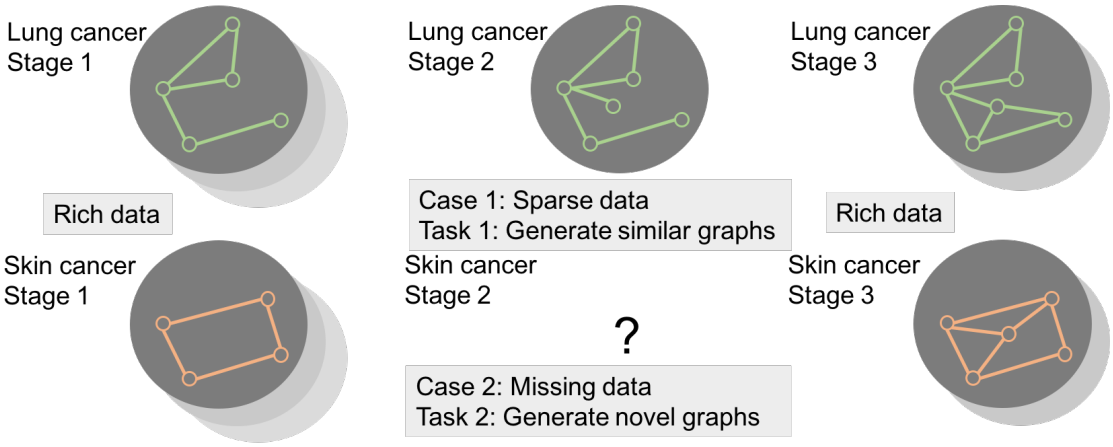


Figure 4.5: Toy example of conditional structure generation: Real-world networks nowadays are often associated with correlated semantic attributes/labels. This allows us to explore the possible correspondence between graph contexts and structures, which can be leveraged to generate structures for graphs with certain semantic contexts that are hardly observed.

particular graph structures. This brings up a natural but challenging question: Can we generate graph structures *w.r.t.* given semantic conditions?

In this work, we propose and study the novel problem of conditional structure generation, whose goal is to learn and generate graph structures under various semantic conditions indicated by contextual attributes or labels in the networks. Figure 4.5 shows a toy example of biomedical networks, where the interactions of certain genes and proteins may follow related but different patterns for individuals with different diseases (*e.g.*, cancers in different body parts and stages). Due to limited observations, network data of some diseases may be more scarce (only one network observed for *Case 1*) or totally missing (no network observed for *Case 2*), while those of other closely related diseases are more available (2-3 networks observed for other cases). Since the diseases are semantically related, their corresponding gene networks may well share certain graph structures. Thus, by efficiently exploring the possible correspondence between network contexts and structures, an ideal model should be able to generate more similar graphs for conditions with scarce observed graphs (*Task 1*), and generate meaningful novel graphs for conditions without any observed graphs (*Task 2*). The problem is important because the generated networks can be valuable in various subsequent studies such as the understanding and prediction of disease development. It is also general, if we consider vast other examples such as in social networks, where users in different communities may share certain connection patterns, and in knowledge graphs, where different types of entities may be related in particular ways.

Existing works on network generation cannot flexibly handle various semantic conditions.

Specifically, earlier probabilistic graph models can only generate networks with limited pre-assumed properties like random links [97], small diameters [98], power-law distribution [99], *etc.*. Recent works based on neural networks can generate graphs with much richer properties learned on given graphs. However, they either only work with single graphs and fixed sets of nodes [101, 102, 103, 104, 105, 106], or model single representative sets of graphs which essentially belong to the same semantic group [107, 108, 109, 110, 111]. Only [110] mentions the ability of conditional generation, but the conditions in their setting are direct graph properties such as number of nodes, which is fundamentally different from the semantic conditions as we consider in this work. Moreover, none of the existing methods really solve the fundamental challenge of graph permutation invariance [205, 168, 206, 207, 208] during their translation between graph structures and representations, due to the facts that their embedding spaces or generated graphs are essentially not permutation-invariant (Sec. 4.5), so they tend to generate different graphs given the same input graphs with permutated node orders (Sec. 4.6).

Thanks to the surge of deep learning [116, 117], many successful neural network models like skip-gram [25] and CNN [209] have been studied for graph representation learning [29, 30, 26, 32, 35]. Among them, graph convolutional neural networks (GCN) [32] has received extensive theoretical analyses and empirical studies recently [205, 168, 33, 34, 210], due to its proved ability to encode nodes, (hyper)links or whole graphs into a permutation-invariant space. However, how to map the distributed vectors back to graphs in a permutation-invariant manner still remains an open problem. Particularly, the graph variational autoencoder (GVAE), as the direct application of GCN for graph generation [101], still only models single networks with fixed sets of nodes (with fixed orders), thus cannot handle flexible semantic conditions and permutation invariance.

In this work, to address the essential challenges of *flexible context-structure conditioning* and *permutation-invariant graph generation* in conditional structure generation, we propose the novel model of CondGen, which is essentially a neural architecture of graph variational generative adversarial nets. It fully leverages the well developed GCN model by further collapsing the node encoding into permutation-invariant graph encoding through variational inference on the conjugate latent distributions, which naturally allows flexible graph context-structure conditioning. To further guarantee permutation-invariant graph decoding/generation, GCN is leveraged again to construct a graph discriminator before the computation of graph reconstruction loss in the standard encoder-decoder framework. This allows the graph generator to explore graphs of variable sizes and arbitrary node orders, which is critical for the capturing of essential graph structures. Finally, for efficient and robust model training, we let the GCNs in graph encoder and discriminator share parameters

to enforce mapping consistency between the graph context and structure spaces and avoid the encoder collapse.

To fully demonstrate the value of conditional structure generation and the power of our proposed CondGen model, we create two benchmark datasets of real-world context-enriched networks and design a series of experiments to evaluate CondGen against several state-of-the-art graph generative models properly adapted to the same setting. Through close comparisons over various graph properties and careful visual inspections, we comprehensively show the supreme effectiveness and generalizability of CondGen on conditional structure generation.

4.5 MODEL: CONDITIONAL GRAPH GENERATION (CONDGEN)

4.5.1 Problem Formulation

We focus on the novel problem of conditional structure generation. We are given a set of graphs $G = \{G_1, G_2, \ldots, G_n\}$, where $G_i = \{V_i, E_i\}$ corresponds to a particular *graph structure* described by the set of nodes V_i and the set of edges E_i. Since graphs nowadays are often contextualized with certain semantic attributes or labels of interest, we construct a condition vector C_i for each graph G_i, which describes some particular simple *graph contexts* of G_i (examples are shown in the data preparation in Sec. 4.6). We leave the exploration of more complex contexts as future work.

In this work, we aim to explore and model the possible context-structure correspondence on graphs. That is, by training a model $\mathcal{M}$ on a set of graphs with certain conditions (*i.e.*, $\mathcal{T} = \{G_i, C_i\}_{i=1}^n$), we hope to (1) given a seen condition $C \in \mathcal{T}$, generate more graphs G mimicking the structures of those in the training set $\mathcal{T}$, and (2) given an unseen condition $C \notin \mathcal{T}$, generate reasonable novel graphs G that can support similar tasks in $\mathcal{T}$ while providing insight into the unobservable world.

We summarize the essential challenge of conditional structure generation as two folds in the following.

Requirement 1. *Flexible context-structure conditioning.* Both the context space, structure space and mapping between the two spaces can be rather complex. Therefore, a model should be able to effectively explore the two spaces and their correspondence based on all context-structure pairs in $\mathcal{T}$. This means the model needs to jointly capture arbitrary contexts and generate graphs of arbitrary sizes and structures. Moreover, a single model has to be trained on arbitrary numbers of context-structure pairs upon availability.

Remark 1. Existing graph generative models only consider graph structures and ignore the rich graph contexts associated for structure generation. Moreover, earlier works only model particular families of structures [211, 212], while more recent works mostly consider single graphs with fixed sizes [101, 102, 103, 104, 105, 106]. GraphRNN [107] is the only one we have seen so far that can be trained with a set of graphs and scale up to graphs with hundreds of nodes, but its GRU design with sequential hidden spaces makes it hard to directly apply effective semantic conditioning (as we will show more details in Sec. 4.6).

Requirement 2. *Permutation-invariant graph generation.* The structure of a graph G is most commonly represented by an adjacency matrix A, where $A_{ij} = 1$ means v_i and v_j are connected and $A_{ij} = 0$ otherwise. However, the representation is not unique. In fact, since there are $n!$ possible permutations for a graph with n nodes, the number of possible adjacency matrices corresponding to the same underlying graph is also exponential. Therefore, a model should be able to efficiently compare the underlying graphs instead of the representations and equalize different representations of the same underlying graphs, essentially achieving permutation-invariance [205, 168, 206, 207, 208].

Remark 2. Existing graph generative models are not permutation-invariant. Particularly, models relying on fixed sets of nodes are not permutation-invariant, because there exists no canonical node ordering and the models have to be re-trained whenever the ordering of nodes is changed [101, 106, 108, 103, 104, 110]. Moreover, models that convert between graphs and other structures like node-edge sequences, trees and random walks are also not permutation-invariant, because there is no guarantee of one-to-one mapping between graphs and the selected structures [107, 109, 102, 111, 105].

4.5.2 Proposed Model

We propose CondGen, which coherently joins the power of GCN, VAE and GAN for conditional structure generation, and satisfies both requirements above. Figure 4.6 illustrates the overall architecture of CondGen. In the following, we introduce the motivations and details of our model design.

Given the two requirements, we get inspiration from recent works on GCN, which is promising in calculating representative and permutation-invariant graph embedding [205, 168]. It is thus natural to think of a permutation-invariant graph encoder-decoder framework by leveraging GCN and enable flexible conditioning through variational inference [213]. In fact, [101] proposed a VAE framework for graph generation soon after the invention of GCN. However, they only consider learning on a single graph $G = \{V, E\}$ and generat-

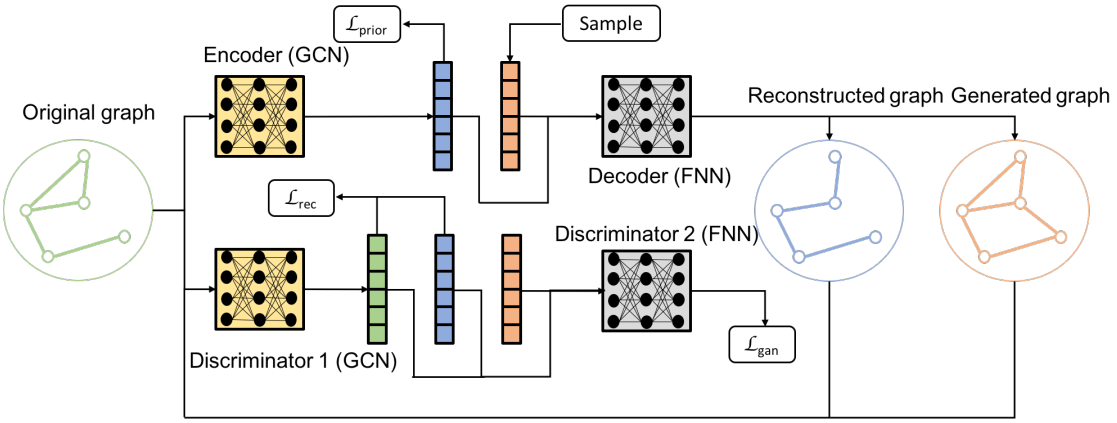


Figure 4.6: Overall framework of CondGen: The upper part is a graph variational autoencoder, where we collapse the node embeddings into a single graph embedding, so as to enable flexible graph context-structure conditioning and allow training/generating of graphs with variable sizes. The lower part makes up for a graph generative adversarial nets, where we leverage GCN to guarantee permutation-invariant graph encoding, generation and comparison for reconstruction. Parameters in the decoder and generator networks as well as those in the two GCN networks in the encoder and discriminator are shared to further boost efficient and robust model inference.

ing/reconstructing links E on the fixed set of nodes V, thus failing to meet both requirements for conditional structure generation.

In this work, we apply a small but necessary trick to the original GVAE framework in [101], *i.e.*, latent space conjugation, which effectively converts node-level encoding into permutation-invariant graph-level encoding, and allows learning on arbitrary numbers of graphs and generation of graphs with variable sizes. Particularly, given a graph $G = \{V, E\}$, since we consider available node contents as semantic conditions, we regard G as a plain network with the adjacency matrix A and generate node features $X = X(A)$ as the standard k-dim spectral embedding[11] based on A. As suggested by reviewers, we later also experiment with replacing spectral embedding by Gaussian random vectors, which leads to significant reduce in runtime and comparable model performance, thanks to the representative and permutation-invariant structure encoding of GCN (details in Sec. 4.6).

Following [101], we introduce the stochastic latent variable Z, which can be inferred from X and A as $q(Z|X, A) = \prod_{i=1}^{n} q(\mathbf{z}_i|X, A)$. $\mathbf{z}_i \in Z$ can be regarded as the *node embedding* of

[11]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html

$v_i \in V$. Different from [101], we use a single distribution $\bar{\mathbf{z}}$ to model all $\mathbf{z}_i$'s by enforcing

$$q(\mathbf{z}_i|X,A) \sim \mathcal{N}(\bar{\mathbf{z}}|\bar{\mu}, \mathrm{diag}(\bar{\sigma}^2)), \text{ where } \bar{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{g}_\mu(X,A)_i, \bar{\sigma}^2 = \frac{1}{n^2}\sum_{i=1}^{n}\mathbf{g}_\sigma(X,A)_i^2, \quad (4.13)$$

where $\mathbf{g}(X,A) = \tilde{A}\mathrm{ReLU}(\tilde{A}XW_0)W_1$ is a two-layer GCN model. $\mathbf{g}_\mu(X,A)$ and $\mathbf{g}_\sigma(X,A)$ compute the matrices of mean and standard deviation vectors, which share the first-layer parameters W_0. $\mathbf{g}(X,A)_i$ is the ith row of $\mathbf{g}(X,A)$. $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the symmetrically normalized adjacency matrix of G, where D is its degree matrix with $D_{ii} = \sum_{j=1}^{n} A_{ij}$.

The trick of latent space conjugation leads to the modeling of $\bar{\mathbf{z}}$, which essentially is the mean of $\mathbf{z}_i$ over G, and thus can be regarded as the *graph embedding* of G. While straightforward, the introduction of $\bar{\mathbf{z}}$ is critical for conditional structure generation, because (1) it allows the model to generate graphs of variable sizes and be trained on set of graphs; (2) it enables graph-level variational inference and flexible context-structure conditioning; (3) it guarantees permutation-invariant graph encoding. We discuss about these three advantages in details in the following.

Firstly, by individually modeling the embedding $\mathbf{z}_i$ of each node $v_i \in V$ with separate latent distributions, [101] can only generate links among the fixed set of nodes V, whereas we can generate graphs of an arbitrary size m by sampling $\mathbf{z}_i$ for m times from the shared distribution of $\bar{\mathbf{z}}$. Secondly, according to [214], a conditional GVAE can be directly constructed by concatenating ($\odot$) the condition vector C to both X and $\bar{\mathbf{z}}$ during training and to $\mathbf{z}_i$'s sampled from $\bar{\mathbf{z}}$ during generation. Finally, since $\mathbf{g}(X,A)$ is permutation-invariant (*i.e.*, $\forall P \in \{0,1\}^{n\times n}$ as a permutation matrix, $\mathbf{g}(PX, PAP^T) = P\mathbf{g}(X,A)P^T$ [154]), $\bar{\mathbf{z}}$, $\bar{\mu}$ and $\bar{\sigma}$ are also permutation-invariant (*i.e.*, $\sum_{i=1}^{n}\mathbf{g}(PX,PAP^T)_i = \sum_{i=1}^{n}[P\mathbf{g}(X,A)P^T]_i = \sum_{i=1}^{n}\mathbf{g}(X,A)_i$). It thus guarantees that $\bar{\mathbf{z}}$ is indistinguishable if A is permutated.

Besides this difference, after sampling a desirable number of $\mathbf{z}_i$'s, to improve the capability of the graph decoder, we append a few layers of fully connected feedforward neural networks $\mathbf{f}$ to $\mathbf{z}_i$ before computing the logistic sigmoid function for link prediction, *i.e.*,

$$p(A|Z) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(A_{ij}|\mathbf{z}_i,\mathbf{z}_j), \text{ with } p(A_{ij}=1|\mathbf{z}_i,\mathbf{z}_j) = \sigma(\mathbf{f}(\mathbf{z}_i)^T\mathbf{f}(\mathbf{z}_j)), \quad (4.14)$$

where $\sigma(z) = 1/(1+e^{-z})$. We optimize the model by minimizing the minus variational lower bound

$$\mathcal{L}_{vae} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{prior}} = \mathbb{E}_{q(Z|X,A)}[\log p(A|Z)] - D_{\text{KL}}(q(Z|X,A)||p(Z)), \quad (4.15)$$

where $\mathcal{L}_{\text{rec}}$ is a link reconstruction loss and $\mathcal{L}_{\text{prior}}$ is a prior loss based on the Kullback-Leibler divergence towards the Gaussian prior $p(Z) = \prod_{i=1}^{n} p(\mathbf{z}_i) = \mathcal{N}(\bar{\mathbf{z}}|\mathbf{0}, \mathbf{I})^n$. The model now consists of a GCN-based graph encoder $\mathcal{E}(A) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{g}(X(A), A)_i$, and an FNN-based graph decoder/generator $\mathcal{G}(Z) = \mathbf{f}(\mathbf{z}_i)^T \mathbf{f}(\mathbf{z}_j)$.

With this modified GVAE, we can compute permutation-invariant graph encoding and generate graphs of variable sizes under different conditions. However, the graph generation process is still not permutation-invariant, because $\mathcal{L}_{rec}$ is computed between the generated adjacency matrix $A' = \mathcal{G}(Z)$ and the original adjacency matrix A, which means A' has to follow the same node ordering as A. In an ideal case, if $A' = PAP^T$, $\mathcal{L}_{rec}$ should be zero. This is not the case for the current model, which misleads the generator/decoder to waste its capacity in capturing the $n!$ node permutations, instead of the underlying graph structures.

To deal with this deficiency, we again leverage GCN, by devising a permutation-invariant graph discriminator, which learns to enforce the intrinsic structural similarity between A' and A under arbitrary node ordering. Particularly, we construct a discriminator $\mathcal{D}$ of a two-layer GCN followed by a two-layer FNN, and jointly train it together with the encoder $\mathcal{E}$ and decoder/generator $\mathcal{G}$, by optimizing the following GAN loss of a two-player minimax game

$$\mathcal{L}_{gan} = \log(\mathcal{D}(A)) + \log(1 - \mathcal{D}(A')), \text{ with } \mathcal{D}(A) = \mathbf{f}'(\mathbf{g}'(X(A), A)), \tag{4.16}$$

where X, $\mathbf{g}'$ and $\mathbf{f}'$ are spectral embedding, GCN and FNN, respectively, similarly as defined before. After $\mathbf{g}'$, the encodings $\mathbf{g}'(A)$ and $\mathbf{g}'(A')$ are permutation-invariant (*i.e.*, $\forall A' = PAP^T, \mathbf{g}'(A) = \mathbf{g}'(A')$), and the reconstruction loss $\mathcal{L}_{rec}$ can be simply computed as $\mathcal{L}_{rec} = ||\mathbf{g}'(A) - \mathbf{g}'(A')||_2^2$, which captures the intrinsic structural difference between A and A' regardless of the possibly different node ordering.

At this point, we find our model closely related to the recently popular framework of VAEGAN [215, 216, 217]. Similarly to their observations, we find it beneficial to include two sources of generated matrix A', *i.e.*, one from the sampled graph encoding Z_s *w.r.t.* the prior distribution, and another from the computed graph encoding $Z_c = \mathcal{E}(A)$, and redefine the GAN loss as

$$\mathcal{L}_{gan} = \log(\mathcal{D}(A)) + \log(1 - \mathcal{D}(\mathcal{G}(Z_s))) + \log(1 - \mathcal{D}(\mathcal{G}(Z_c))). \tag{4.17}$$

Different from VAEGAN, and motivated by the powerful framework of CycleGAN [218], we further aim to apply additional constraints to the framework to enforce mapping consistency between the context and structure spaces. Particularly, we find it beneficial to share

parameters in the two GCN modules $\mathbf{g}$ and $\mathbf{g}'$, which essentially requires that the generated graph A' can be brought back to the latent space of graph encoding with contexts $Z \odot C$ by the same encoder $\mathbf{g}$ that maps the original graph A to the space of $Z \odot C$. Besides, in practice, it may also help prevent the encoder from occasional collapse due to the overwhelmingly powerful decoder/generator [219], when $\mathcal{E}$ keeps yielding the same noise Z for different input A, but $\mathcal{G}$ manages to overfit the training data by generating the correct A' solely based on the condition vector C. In this case, the model degrades into a conditional GAN [214], which is harder to train without $\mathcal{E}$ functioning as expected.

4.5.3 Training Details

We jointly train the encoder $\mathcal{E}$, decoder/generator $\mathcal{G}$ and discriminator $\mathcal{D}$ by optimizing the following combined loss function

$$\mathcal{L}_{\textsc{CondGen}} = \mathcal{L}_{rec} + \lambda_1 \mathcal{L}_{prior} + \lambda_2 \mathcal{L}_{gan}, \tag{4.18}$$

where λ_1 and λ_2 are tunable trade-off hyperparameters. As suggested in [215], it is important not to update all model parameters *w.r.t.* the combined loss function. Particularly, we use the following parameter updating rules for in each training batch

$$\theta_E \overset{+}{\leftarrow} -\nabla_{\theta_E}(\mathcal{L}_{rec} + \lambda_1 \mathcal{L}_{prior}),\ \theta_G \overset{+}{\leftarrow} -\nabla_{\theta_G}(\mathcal{L}_{rec} - \lambda_2 \mathcal{L}_{gan}),\ \theta_D \overset{+}{\leftarrow} -\nabla_{\theta_D}\lambda_2 \mathcal{L}_{gan}. \tag{4.19}$$

4.6 EXPERIMENTAL EVALUATIONS

We create two real-world context-rich network datasets and conduct thorough experiments to demonstrate the effectiveness and generalizability of CondGen in conditional structure learning. All code and data used in our experiments have been made available on GitHub[12].

Datasets. Since we are the first to consider the novel but important problem of conditional structure generation, there is no existing dataset for evaluation. To this end, we created two benchmark datasets, *i.e.*, a set of author citation networks from DBLP[13] and a set of gene interaction networks from TCGA[14].

From DBLP, we create a set of 72 ($8 \times 3 \times 3$) author networks, each associated with a 10-dim condition vector. The nodes are the first authors of research papers published in

[12]https://github.com/KelestZ/CondGen
[13]DBLP source: https://dblp.uni-trier.de/
[14]TCGA source: https://www.cancer.gov/tcga

8 conferences, *i.e.*, NIPS and ICML (representing the ML community), KDD and ICDM (DM), SIGIR and CIKM (IR), SIGMOD and VLDB (DB). Then each of the 8 groups of authors are further divided into 3 subgroups by the number of total publications (1-10, 10-30, 30+), representing the productivity of authors. Finally three networks are created for each of the 24 sets of authors, by adding in the citation links created in different time period (1990-1999, 2000-2009, 2010-2019). Thus, the 10-dim condition vector is a concatenation of a 8-dim one-hot vector denoting the conferences, and a 2-dim integral vector denoting the level of productivity and link creation time (each with three values 0, 1, 2). The average numbers of nodes and edges in the author networks are 109 and 186, respectively.

From TCGA, we create a set of 54 ($6 \times 3 \times 3$) gene networks, each associated with a 8-dim condition vector consisting of a 6-dim one-hot encoding of cancer primary sites (brain, liver, lung, ovary, skin, and kidney) and a 2-dim integral vector denoting age of diagnosis (30-57, 58-69, 70-90) and stage approximated by days-to-days (0-400, 400-800, 800-8000). For each faceted search with a particular combination of primary site, age-at-diagnosis, and days-to-death filters, a gene correlation network was created using a gene expression matrix constructed from the first 10 RNA-Seq FPKM files. From each RNA-Seq FPKM matrix M, a transformed matrix $N = log_{10}(M + 0.5 \times min(M))$ was created and then filtered for genes with a unique entrez ID and vector representation [220]. Finally, a gene correlation network was constructed using pearson correlation with p-value threshold 0.01. The average numbers of nodes and edges in the gene networks are 177 and 1096, respectively.

Baselines. Since no baseline is available for the novel task of conditional structure learning, we carefully adapt three state-of-the-art graph generation methods, *i.e.*, GVAE [101], NetGAN [102] and GraphRNN [107], by concatenating the condition vectors to both the node features of the input graph and the output of the last encoding layer following the standard practice in [214]. To allow a single GVAE or NetGAN model to be trained on a set of graphs, we fix the size of input and output graphs as the largest size of all networks following [108]. As suggested by reviewers, we also construct a variant of CondGen by replacing the spectral embedding with Gaussian random vectors of the same sizes to use as input node features to GCN, denoted as CondGen(R) (*i.e.*, random vectors) as opposed to CondGen(S) (*i.e.*, spectral embeddings).

Protocols. To demonstrate the effectiveness and generalizability of CondGen, we evaluate both tasks of mimicking similar seen graphs and creating novel unseen graphs. We firstly partition all networks at random by a ratio of 1:1 into training and testing sets. Note that, the testing set includes graphs with both seen and unseen conditions in the training set, so

a good model that performs well on the testing set has to effectively capture the context-structure correspondence among graphs with the seen conditions and generalize to graphs with unseen conditions.

Parameter Settings. As mentioned, our CondGen model consists of an encoder, a decoder, and a discriminator.

In the encoder, we use a spectral embedding layer to extract the node features solely based on graph structures. The output of the spectral embedding layer is a $n \times d$, where d is set to 5 on DBLP and 10 on TCGA. We select d as such small values because there are some small graphs especially in the DBLP dataset and the Laplacian eigenvectors corresponding to the first few smallest eigenvalues usually capture the most important graph properties such as number of disconnected components, clustering structures, *etc.*. A graph convolution layer follows afterwards with the output size of 16. We notice that simply using graph convolution layers tends to give unstable outputs, so we add two linear layers with a one-dimensional batch normalization layer and a ReLU activation layer before obtaining the mean and variance variables. Both mean and variance vectors have a dimension of 6.

In the decoder, we use a graph convolution layer followed by linear layers. We follow the same design of GVAE to reconstruct graphs, *i.e.*, using the encoded vectors generated from the linear layers multiplied by their transpose vectors. Interestingly, we notice that if the dimension of the encoded vectors is large, the output graphs tend be very dense, while a small dimension may lead the graphs having many disconnected components. Thus the selection of 6 is done through vast cross-validation. However, since the set of candidate values is relatively small (we conduct cross-validation on values of 2-10), the hyperparameter selection process is easy to complete.

The discriminator has similar settings as the encoder, *i.e.*, they share the exact same GCN module followed by FNNs with the same design, except that the output here is a single value, differentiating generated graphs from real graphs.

We use Adam optimizers for the training of all modules in the CondGen with a learning rate of 0.001.

Performances. Following existing works on generative models [102, 107, 108], we evaluate the generated graphs through visual inspection and graph property comparison[15]. Our model can flexibly generate graphs with arbitrary numbers of nodes and edges. For fair and clear comparison, when generating each graph, we set the maximum number of nodes

[15]Statistics we use include LCC (size of largest connected component), TC (triangle count), CPL (characteristic path length), MD (maximum node degree) and GINI (gini index), measuring different properties of graphs.

and edges to the same as the real graph for all compared algorithms. As shown in Table 4.4, the suite of statistics we use measure graphs from different perspectives, and different algorithms often excel at particular ones. Our proposed CondGen models constantly rank top with very few exceptions on all measures over both datasets. The advantage of CondGenon generating graphs with seen conditions in the training set demonstrates its utility in generating more similar graphs for conditions where observations might be sparse, while the edge on unseen conditions indicates its generalizability to semantically relevant conditions where observations are completely missing. The CondGen(R) model variant has quite competitive performance with CondGen(S), which can be explained by the representative and permutation-invariant structure encoding power of GCN. Due to space limit, we put detailed parameter settings, qualitative visual inspections and in-depth model analyses into the appendix in the supplemental materials.

Runtimes. Similar to most neural network models, it is meaningless to compute the exact complexity of CondGen, because the actual runtimes mostly depend on the number of training iterations until convergence. To this end, we record the average runtimes for the training of all compared algorithms until convergence on the two sets of networks and present in Table 4.5. As we can clearly observe, state-of-the-art graph generation algorithms like GraphRNN and NetGAN are rather slow, due to the heavy model of RNN and large number of sampled walks, respectively, while CondGen and its base model GVAE are much faster. Since CondGen and GVAE are basically a simple GCN model encapsulated in a VAEGAN and VAE framework, respectively, we also find that the memory consumptions of CondGen and GVAE are orders of magnitudes lower than GraphRNN and NetGAN. Among the two CondGen variants, CondGen(S) takes about double runtime as CondGen(R), due to the computation of spectral embeddings. While the overhead is not significant, it can get more concerning as the networks become larger, due to the essential $O(n^3)$ complexity of spectral embedding. However, since CondGen(R) has quite competitive performance with CondGen(S), one can use it as a substitute of CondGen(S) when efficiency is more of a concern.

In-depth Model Analyses To understand how our proposed CondGen model learns to capture the key properties of graphs, we closely evaluate it along training. Since the results are averaged among all networks in the dataset, which exhibits various graph structures, the variances are pretty large and often do not cancel with each other. Interestingly, we find that most graph properties tend to have larger values on real graphs than random graphs, and thus an untrained model often gives lower values on them compared with a well trained

model. Nonetheless, CondGen manages to approach the values of real graphs rapidly after around one hundred of epochs on most graphs.

Figure 4.7 shows the in-depth model analyses results on the DBLP dataset, while the results on the TCGA dataset follow the similar trends and are thus omitted. Interested readers are encouraged to run our models which are submitted together in the supplementary materials and see how different models behave during training on the novel task of conditional structure generation. Meanwhile, in order to better demonstrate how the generated graphs can be useful in downstream applications, we are conducting more experiments with advanced graph classification and regression tasks, hoping to see that the graphs generated by CondGen can successfully 'fool' the classification and regression models, providing unlimited structural data under particular conditions of interest that are close to hardly observed or unobservable real graphs.

Graphs	Models	LCC	TC	CPL	MD	GINI
DBLP Seen	Real	96.00	48.54	3.696	11.62	0.3293
	GVAE	20.91**	21.76**	1.390*	2.32**	0.1964**
	NetGAN	21.15**	22.46**	1.641**	2.77**	**0.0568****
	GraphRNN	6.88*	69.32**	1.628**	7.06**	0.2446**
	CondGen(R)	6.70*	**7.70***	1.201*	**1.33**	0.1232*
	CondGen(S)	**6.00**	11.32	**0.963**	1.48	0.0959
DBLP Unseen	Real	102.50	58.21	4.982	14.29	0.3223
	GVAE	17.40**	17.02**	1.521**	3.53*	0.2479**
	NetGAN	29.57**	39.85**	1.494**	3.71**	**0.0812**
	GraphRNN	6.43	73.21**	1.305*	6.43**	0.1447**
	CondGen(R)	9.25*	10.50	1.445**	**1.92**	0.1418**
	CondGen(S)	**6.33**	**10.17**	**1.162**	**1.92**	0.0861
TCGA Seen	Real	177.34	8913.20	4.171	38.27	0.4192
	GVAE	54.82**	2396.94*	1.538	14.10**	0.2035**
	NetGAN	32.02**	3614.61**	1.702**	17.61**	0.1289*
	GraphRNN	**16.20***	2881.68**	1.899**	18.78**	0.2726**
	CondGen(R)	34.42**	2594.16**	1.542	9.50	0.1509**
	CondGen(S)	23.72	**2076.05**	**1.524**	**8.32**	**0.1093**
TCGA Unseen	Real	177.91	8053.18	4.143	34.34	0.4154
	GVAE	37.18**	2768.55**	**1.324***	13.03**	0.1497**
	NetGAN	31.36**	3557.91**	1.645*	18.45**	0.1277**
	GraphRNN	**15.73****	2605.73**	1.859**	13.55**	0.2647**
	CondGen(R)	27.77*	3083.81**	1.362*	10.86*	0.1413**
	CondGen(S)	23.97	**2058.95**	1.522	**8.68**	**0.1003**

Table 4.4: Performance evaluation over compared algorithms regarding several important graph statistical properties. The Real rows include the values of real graphs, while the rest are the *absolute values of differences* between graphs generated by each algorithm and the real graphs. Therefore, smaller values indicate higher similarities to the real graphs, thus better overall performance. We conduct paired t-test between each baseline and CondGen(S), scores with * and ** passed the significance tests with $p = 0.05$ and $p = 0.01$, respectively.

Graphs	GVAE	NetGAN	GraphRNN	CondGen(R)	CondGen(S)
DBLP	12.8	398.6	299.5	31.5	72.3
TCGA	10.9	414.0	192.4	27.6	52.1

Table 4.5: Runtimes of training all compared algorithms on the two sets of networks (minutes).

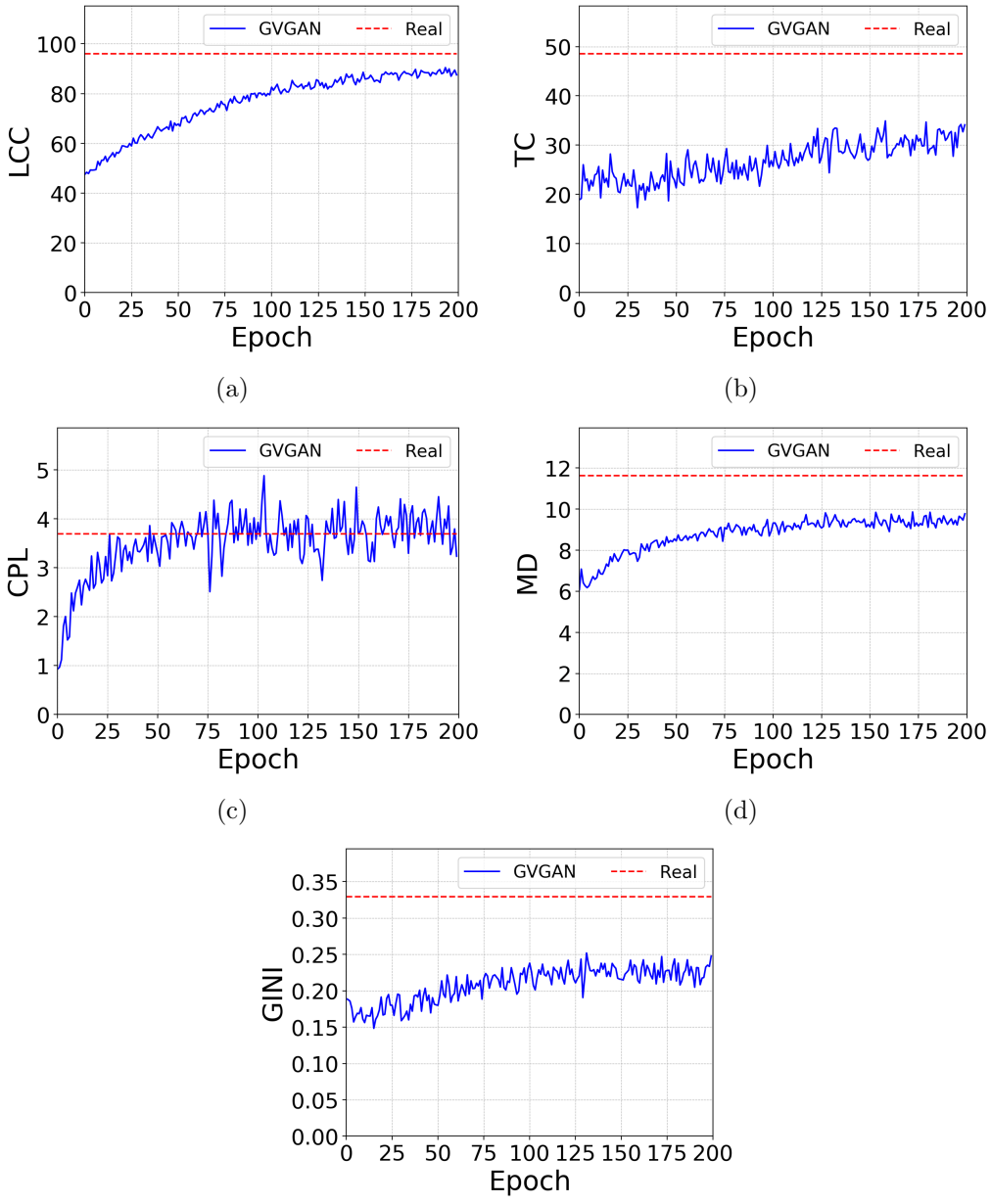


Figure 4.7: Different graph statistics evaluated along the training of CondGen on DBLP (averaged between seen and unseen conditions). CondGen efficiently learns to capture the key properties of graphs and converges to the values of real graphs with only around 100 epochs of training.

CHAPTER 5: APPLICATIONS

In this Chapter, I give two examples of real-world graph mining applications enabled by my proposed framework of contextualized projections, *i.e.*, web-scale recommendation with explicit contexts (Section 5.1-5.3) and new user churn prediction with implicit contexts (Section 5.4-5.5).

5.1 WEB-SCALE RECOMMENDATION WITH EXPLICIT CONTEXT

Recent research on extracting information from large networks (graphs) has been largely focused on graph neural networks [221],

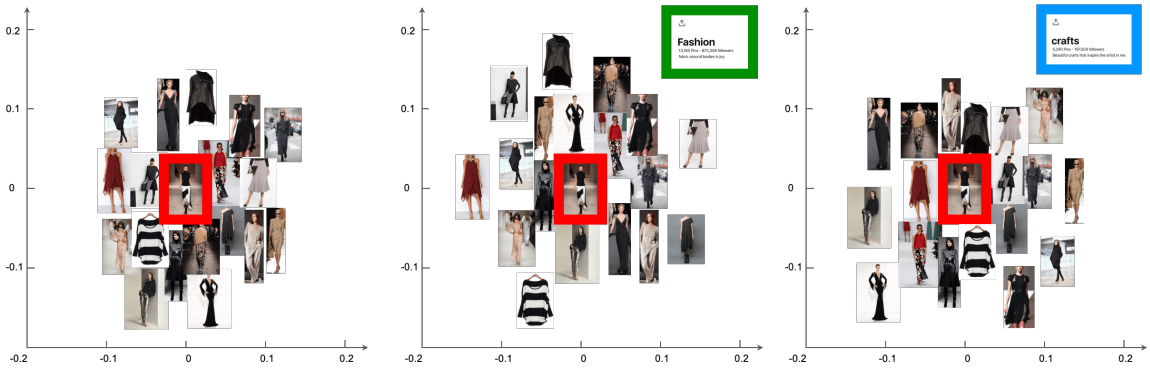
among which, graph convolutional networks (GCNs) have received significant attention [32, 34, 210, 222, 223, 224, 225]. This is not only because of their fundamental connections to spectral graph theory and thus provable representation power [168, 226, 227], but also their promising performance on several graph mining benchmarks [33, 154, 35].

To harness the power of GCNs, GraphSage was designed to enable batch-wise training through fixed-sized neighborhood sampling [33]. It was later adapted to a more robust enterprise-scale version called PinSage [228] and deployed at Pinterest. PinSage was shown to be extremely effective at recommending similar pins based on the industry-scale pin-board graph (billions of nodes with each node having thousands of features).

However, one key limitation of existing GCNs is that they cannot distinguish multi-facet node properties and complex node interactions, which manifests due to their homogeneous treatment of node links. As illustrated in Figure 5.1(a), in real-world industrial platforms like Pinterest, state-of-the-art GCN models mix all related nodes in a single embedding space.

Present work. Here we argue that nodes in a network are connected due to different reasons and are thus *close in different ways*, which cannot be simultaneously captured by a single embedding. To this end, we propose MultiSage, which is based on a novel idea of *contextualized multi-embedding*, where we compute multiple embeddings for network nodes to capture their *contextualized interaction* in the corresponding *multiple embedding spaces*. Figure 5.1(b) illustrates the scenario where MultiSageretrieves and organizes nodes related to the query under different contexts in multiple embedding spaces. Our MultiSageanswers two important questions: (1) how to find proper context; And (2) how to leverage context in massive real-world networks to facilitate effective and flexible downstream applications.

RQ 1: How to find proper context? Real-world applications often care most about par-



(a) PinSage Global Single-Embedding

(b) MULTISAGE Contextualized Multi-Embedding

Figure 5.1: A toy example of related pin recommendation in Pinterest. In Subfigure (a), given the query pin (in red), PinSage computes a single pin embedding and mixes up all related pins. On the contrary, in Subfigure (b), we compute multiple pin embeddings based on different boards (*e.g.*, fashion and crafts), which naturally organizes related pins according to their contextualized distances to the query and effectively distinguishes relatedness in different perspectives (fashion models are drawn towards the query in the first space, whereas craft clothes in the second).

ticular types of nodes (*e.g.*, papers in academic graphs, users in social networks, *etc.* [228, 229, 230]). However, we observe that real-world networks are often multipartite, *i.e.*, including multiple types of nodes, which naturally provide the context of interactions for each other. Due to this observation, we find it beneficial to model *target nodes* and *context nodes* in multipartite networks, where the interactions among target nodes can be subtly modeled under the help of context nodes.

Take Pinterest as an example, where users interact with *pins* (*e.g.*, images in Figure 5.1) mostly by pinning them to customized *boards*, thus creating a massive bipartite pin-board graph. Since the embeddings of pins are critical for various downstream services like search and recommendation, we regard all pins as the target nodes and aim to compute high-quality contextualized multi-embeddings for them. In the meantime, we regard each board node as a context node and leverage the fact that a board provides context for the relationship between two pins.

The intuition behind leveraging boards to contextualize interactions among pins is natural—for example, if the paths connecting two particular pins mostly pass through a fashion board, the embeddings of the two pins are likely close because they both describe fashionable items. Besides simplicity, we also find the idea general—for example, on a publication network like

OAG,[1] if two papers are mostly connected by paths passing through a data mining venue, they are likely close because they both study data mining problems.

RQ 2: How to leverage context? We propose MultiSage, which leverages ubiquitous graph context in real-world multipartite networks to empower GCN by injecting interaction contextualization into its critical neighborhood convolution process, where we dynamically compute multiple embeddings for each target node under the conditions implied by different context nodes. Particularly, we design a novel GCN architecture with a learnable contextual masking operation based on context node features for flexible feature-level embedding projection, and a three-way contextual attention mechanism for node-level neighbor reweighing during graph convolutions. To fully capture the rich information in web-scale networks, we further implement a parallel contextualized random walk engine and an efficient Hadoop2-based data provider pipeline to pre-join and dynamically feed training data to the multi-GPU model trainer, which allows scalable model training on massive networks with millions to billions of nodes.

We conduct extensive experiments and case studies on an enterprise Pinterest pin-board network as well as a public OAG publication network. The advantages of MultiSage are intriguing not only because it outperforms various state-of-the-art baselines with significant margins (9%-25% on MRR over the production model of PinSage) by incorporating rich and subtle network information, but also due to its corroborated utility in generating flexible and meaningful multi-embeddings that naturally paves the way to fine-grained search and recommendation.

5.2 MODEL: GRAPHSAGE WITH CONTEXTUALIZED MULTI-EMBEDDING (MULTISAGE)

5.2.1 Preliminaries

Following abundant recent works on GCN [33, 34, **?**, 224], we take [32] proposed by Kipf and Welling as a representative to briefly recapitulate its main design. Particularly, the typical output of the $(l+1)$-th convolutional layer $\mathbf{H}^{(l+1)}$ of GCN is computed as follows

$$\mathbf{H}^{(l+1)} = \sigma\Big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\Big), \tag{5.1}$$

where $\tilde{\mathbf{A}}$ is the adjacency matrix with self-connections of the whole graph with N nodes, $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, $\mathbf{W}^{(l)}$ is the trainable layer-wise weight matrix, and $\sigma(\cdot)$ is a nonlinear acti-

[1] https://www.openacademic.ai/oag/

vation function such as ReLU. $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times D_l}$ is the output of the l-th layer, with $\mathbf{H}^{(0)} = \mathbf{X}$, *i.e.*, the original node features.

One major drawback of [32] is the requirement of putting the whole graph (*i.e.*, $\tilde{\mathbf{A}}, \tilde{\mathbf{D}} \in \mathbb{R}^{N \times N}$) into the main memory (or GPU memory), which limits the training to graphs with only thousands of nodes. To address this issue, GraphSage [33] is proposed to sample a fixed number of neighbors in each convolution layer and aggregate the neighborhood embedding as follows

$$\mathbf{h}_{\mathcal{N}(v)}^{(l+1)} = \text{AGGREGATE}(\{\mathbf{h}_u^{(l)}, \forall u \in \mathcal{N}(v)\}), \tag{5.2}$$

where $\mathcal{N}(v)$ is the sampled neighborhood of node v, and AGGREGATE is the aggregation function such as mean pooling.

To fully leverage the model capacity of GCN and scalability of GraphSage, PinSage [228] is developed at Pinterest for the particular task of related pin recommendation. To suit this real-world recommendation task, a series of techniques are adopted, while the major one lies in the triplet-wise optimization objective based on max-margin ranking as follows

$$\mathcal{J}(v_q, v_p, v_n) = \max\{0, \mathbf{h}_{v_q}^T \mathbf{h}_{v_n} - \mathbf{h}_{v_q}^T \mathbf{h}_{v_p}^L + \delta\}, \tag{5.3}$$

where δ is a margin hyper-parameter. In each triplet (v_q, v_p, v_n), v_q and v_p are the *query* and *positive* nodes sampled from available training data (*e.g.*, related pin pairs generated from users' interactions with pins), while v_n is the *negative* node sampled from $P_n(v_q)$ (*i.e.*, the distribution of negative examples for v_q).

5.2.2 Problem Definition

In this work, we propose to leverage the heterogeneity of real-world networks by separating different types of nodes into *target nodes* and *context nodes* based on application need, so as to focus the computations on the important types of nodes, while enabling flexible contextualization based on the others.

Figure 5.2 gives an example of separating the multipartite Pinterest network into target nodes (*i.e.*, pins) and context nodes (*i.e.*, boards), where pins are related via boards, which naturally describe their interaction contexts. Note that, we make two assumptions to achieve such desired simplifications of the otherwise complicated heterogeneous networks: (1) minimum domain knowledge is available to separate target nodes from context nodes; (2) most important interactions among target nodes involve context nodes. To show that both

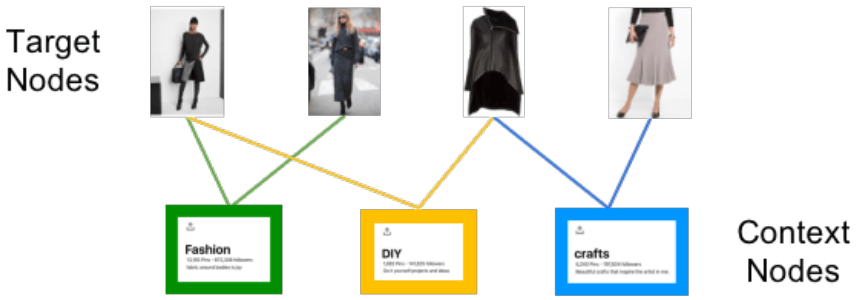


Figure 5.2: Target-context separation of Pinterest network.

assumptions are general and realistic, we give examples of a few commonly used multipartite networks in Table 5.1. Note that, while we focus most of our discussion on the major context nodes, the framework we develop is easily extendable to incorporate various other context nodes.

Dataset	Target node	Major context	Other context
IMDB [7]	movie	genre	director, actor
TCGA [9]	gene	pathway	disease, species
OAG [113]	paper	venue	author, keyword
Pinterest [112]	pin	board	user, session

Table 5.1: Target and context nodes on different networks.

Under different conditions characterized by context nodes, proximity among target nodes can be rather different. For example, a fashionable pair of glasses and a fashionable pair of shoes are close under fashion and far away under crafts. In this work, to capture the complex multi-perspective interactions among network nodes, we propose to improve the successful GCN models with the novel computation of contextualized multi-embedding as follows

Definition 4 *Contextualized Multi-Embedding. Given a network with target nodes $\mathcal{T}$ and context nodes $\mathcal{C}$, compute $|\mathcal{C}|$ embeddings for each of the $|\mathcal{T}|$ nodes, so that the conditional proximity $p(v, u|o)$ between any two target nodes $v, u \in \mathcal{T}$ under the condition indicated by any context node $o \in \mathcal{C}$ is captured in the embedding space corresponding to o.*

5.2.3 MultiSage

Contextualized graph convolution. Figure 5.3 illustrates the detailed architecture of MultiSage. Based on GCN [32], MultiSage leverages graph convolutional layers to

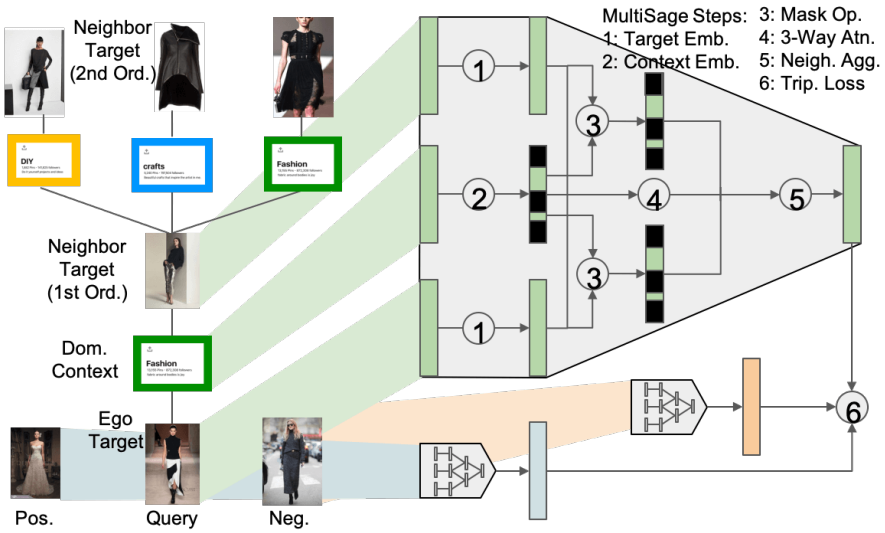


Figure 5.3: The overall neural architecture of MultiSage.

generate embeddings for each target node by treating it as the *ego* and aggregating information from its *neighbor* on the graph. However, existing GCNs do not leverage context nodes during graph convolution, and apply the same aggregation function to all neighbors by treating them equally. As we stress in this work, every ego target node interacts with its neighbor target node under different conditions characterized by the intermediate context nodes, so neighbors with different contexts should be distinguished during aggregation.

Take Figure 5.4 as an example. In (a), when three neighbor pins (*i.e.*, target nodes) are aggregated through mean pooling, the resulting neighborhood embedding simply lies in the center of the three pins, reflecting the same influence of all neighbors on the ego. As a contrast in (b), two neighbor pins are connected via the fashion board (*i.e.*, context node) while the other one via the crafts board (*i.e.*, another context node), thus drawing the neighborhood embedding more into the fashion direction. Such contextualization over the target interaction is desirable, since each neighbor is similar to the ego from a particular perspective, and thus should influence the ego embedding more in the corresponding subspace.

To achieve such interaction contextualization, we firstly retrieve the *dominant* context node between each pair of ego and neighbor target nodes with a parallel contextualized random walk engine (details deferred to Section 5.2.3.2). As a result, each neighbor $u \in \mathcal{N}_v \subset \mathcal{T}$ of the ego $v \in \mathcal{T}$ is associated with a dominant context node $o \sim (v, u) \in \mathcal{C}$ that characterizes the interaction between v and u. Based on this, in the following we describe how to learn the target embedding $\mathbf{z}_t$ for both v and u, and the context embedding $\mathbf{z}_c$ for o, and combining $\mathbf{z}_t$ and $\mathbf{z}_c$ for *contextualized graph convolution*.

Raw feature transformation. Since $|\mathcal{T}|$ and $|\mathcal{C}|$ can both be very large in real-world

networks (*e.g.*, millions), identity based embedding is impractical. To this end, we adopt the common practice of feature based embedding for both target and context nodes, which learns to project and transform raw node features via stacked dense neural networks as follows (Steps 1-2 in Figure 5.3)

$$\begin{aligned}\mathbf{z}_t &= \mathrm{ReLU}\big(\mathbf{W}_t^{(K)} \dots \mathrm{ReLU}(\mathbf{W}_t^{(1)}\mathbf{x}_t + \mathbf{b}_t^{(1)}) \dots + \mathbf{b}_t^{(K)}\big),\\ \mathbf{z}_c &= \mathrm{ReLU}\big(\mathbf{W}_c^{(K)} \dots \mathrm{ReLU}(\mathbf{W}_c^{(1)}\mathbf{x}_c + \mathbf{b}_c^{(1)}) \dots + \mathbf{b}_c^{(K)}\big),\end{aligned} \tag{5.4}$$

where $\mathbf{x}_t$ and $\mathbf{x}_c$ are the raw features of target and context nodes, respectively, and $\big\{\mathbf{W}_t^{(k)}, \mathbf{b}_t^{(k)}, \mathbf{W}_c^{(k)}, \mathbf{b}_c^{(k)}, \forall k \in \{1, \dots, K\}\big\}$ are the learnable parameters of the target and context embedding layers shared on the whole graph, which are independent of the graph structures and sizes ($|\mathcal{T}|$ and $|\mathcal{C}|$). $\mathbf{z}_t$ and $\mathbf{z}_c$ are the embeddings of target and context nodes in the shared embedding space.

Contextual masking. The next step is to transform and aggregate target embeddings $\mathbf{z}_t$ based on context embeddings $\mathbf{z}_c$. Motivated by the example in Figure 5.4, we design and apply a *contextual masking operation* by setting the size of $\mathbf{z}_t$ and $\mathbf{z}_c$ to be the same, and then element-wise multiplying the embedding of dominant context node $\mathbf{z}_c(o)$ onto the embeddings of both ego and neighbor target nodes $\mathbf{z}_t(v)$ and $\mathbf{z}_t(u)$ as follows (Step 3 in Figure 5.3)

$$\mathbf{z}_{t|c} = \mathbf{z}_t \otimes \mathbf{z}_c. \tag{5.5}$$

Note that, since the last embedding layer of $\mathbf{z}_c$ is ReLU, certain dimensions can be learned to be zero, which effectively "masks out" irrelevant dimensions, so as to project the target embeddings into particular subspaces directly controlled by the context embeddings, which exactly matches the geometric intuitions in Figure 5.4.

Although the contextual masking operation is geometrically intuitive, to be comprehensive, we also explore and employ other schemes to compute $\mathbf{z}_{t|c}$. Motivated by the popular translation based relational models [67, 68], we compute $\mathbf{z}_{t|c} = \mathbf{z}_t \oplus \mathbf{z}_c$, which "translates" target embedding into different spaces by element-wise summation with context embedding. Moreover, motivated by the power of dense neural networks [231], we also attach freely learnable dense neural networks to the concatenation of target and context embedding, *i.e.*, $\mathbf{z}_{t|c} = \mathrm{ReLU}\big(\mathbf{W}_p(\mathbf{z}_t \odot \mathbf{z}_c) + \mathbf{b}_p\big)$.

Contextual attention. Contextual masking allows us to project different ego-neighbor pairs into various embedding subspaces indicated by the dominant context, so as to emphasize contextualized interaction among target node pairs regarding particular embedding

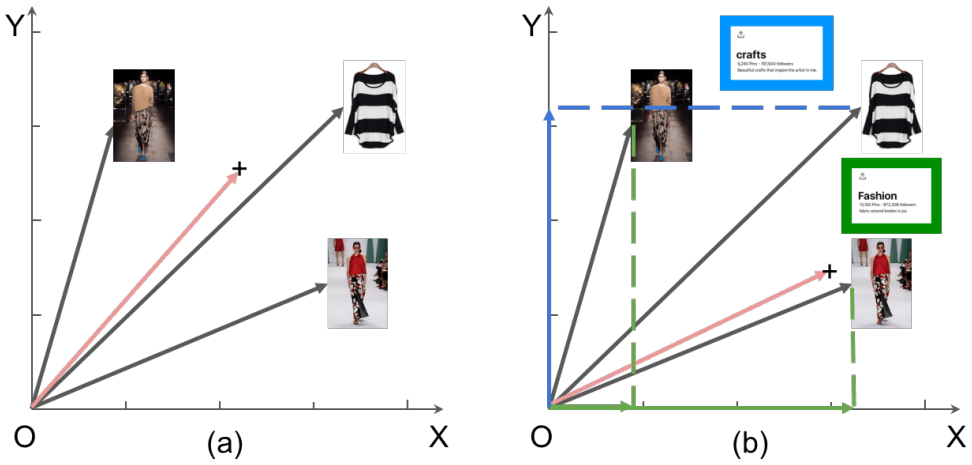


Figure 5.4: Geometrics of contextualized multi-embedding.

dimensions at the *feature level* during graph convolutions. However, it does not consider the overall impact of different neighbors on particular egos at the *node level*, especially under the consideration of different interaction context.

Motivated by the powerful attention networks [232, 35], we design a novel contextual attention mechanism, by jointly computing an attention weight for each ego-context-neighbor triplet as follows (Step 4 in Figure 5.3)

$$\alpha(v,o,u) = \frac{\exp\Big(\tau\big(\mathbf{a}^T[\mathbf{W}_{at}\mathbf{z}_t(v) \odot \mathbf{W}_{ac}\mathbf{z}_c(o) \odot \mathbf{W}_{at}\mathbf{z}_t(u)]\big)\Big)}{\sum_{u' \in \mathcal{N}_v, o' \sim (v,u')} \exp\Big(\tau\big(\mathbf{a}^T[\mathbf{W}_{at}\mathbf{z}_t(v) \odot \mathbf{W}_{ac}\mathbf{z}_t(o') \odot \mathbf{W}_{at}\mathbf{z}_t(u')]\big)\Big)}, \tag{5.6}$$

where $\{\mathbf{a}, \mathbf{W}_{at}, \mathbf{W}_{ac}\}$ are the learnable attention parameters ($\mathbf{W}_{at}$ for target embedding and $\mathbf{W}_{ac}$ for context embedding), and τ is the LeakyReLU activation function following [233, 35]. $\alpha(v,o,u)$ is learned to assign different weights to neighbors based on the embedding of both context o and target pair (v,u), so as to allow graph convolution at each ego to raise attention to important neighbors and contexts. To further improve the capacity and stability contextual attention, we exploit multi-head attention [232, 35] to compute the aggregated contextualized embedding as follows

$$\mathbf{z}_{\mathcal{N}_v}(x) = \sigma\Big(\frac{1}{D}\sum_{d=1}^{D}\sum_{u \in \mathcal{N}_v, o \sim (v,u)} \alpha^{(d)}(v,o,u)\mathbf{z}_{t|c}(x,o)\Big) \tag{5.7}$$

where each $\alpha^{(d)}$ is a single attention weight computed by Eq. 6, and σ is the Sigmoid function.

x can be either v or u.

While being expressive, our multi-head contextual attention mechanism is also feature based and cheap to compute. Particularly, all attention parameters $\{\mathbf{a}^{(d)}, \mathbf{W}_{at}^{(d)}, \mathbf{W}_{ac}^{(d)}, d \in \{1, \ldots, D\}\}$ are shared across all links in the graph and independent of graph structures and sizes.

Training objective. For MultiSage, we find directly adding up the contextualized ego embedding $\mathbf{z}_{\mathcal{N}_v}(v)$ and neighbor embedding $\mathbf{z}_{\mathcal{N}_v}(u)$ as the final MultiSageembedding $\mathbf{h}(v)$ to be most efficient and beneficial (Step 5 in Figure 5.3). As for the training loss, we follow PinSage to take query-positive pairs (v_q, v_p) from the training data, and apply the hard negative sampling strategy [234, 228] to generate query-positive-negative triplets (v_q, v_p, v_n). After that, we apply the same convolution procedure to v_q, v_p and v_n to compute their MultiSageembedding $\mathbf{h}(v_q)$, $\mathbf{h}(v_p)$ and $\mathbf{h}(v_n)$, which are then optimized with the loss function in Eq. 5.3 (Step 6 in Figure 5.3).

Note that, during graph convolution, the contextualized multi-embeddings are aggregated into a single MultiSageembedding, to be trained and evaluated towards a single node similarity (*e.g.*, pin/paper relatedness, as in Sec 3.2). Nonetheless, we can easily compute the feature-based contextualized multi-embeddings based on the trained model and use them towards fined-grained downstream applications (*e.g.*, contextualized recommendation, as in Sec 3.3).

Web-scale implementation. Now we consider the implementation of MultiSage on real-world web-scale multipartite networks. As an example, in Pinterest, we construct a relatively smaller graph of 76 million pins, 15 million boards, and 2.7 billion links for the rapid development and evaluation of MultiSage, while a larger graph includes 1.4 billion pins, 1.3 billion boards, and 23 billion links. In contrast, most state-of-the-art graph embedding models are only scalable to thousands of nodes and links [32, 34, 225, 35, 63], which are far from useful for real-world large-scale applications. In this work, we develop a series of techniques to scale up MultiSage.

Parallel contextualized random walk. The first complexity of MultiSage lies in the multi-layer graph convolutions, which requires the retrieval of multi-order neighbors on the graph during training. Since neighbors of a node can be almost anywhere on the graph due to the small world phenomenon [98], most GCN-based models simply put the whole adjacency matrix in memory [32, 35], which is impossible for large-scale applications. Another way is to compute minibatches with fixed numbers of neighbors [33], but such minibatches still easily become too large and complex when the convolution depth increases.

To simplify the multi-layer architecture of GCN while maintaining its long-range propaga-

tion ability, we get motivated by [225], which shows that a single-layer GCN with neighbors sampled according to personalized PageRank scores can mimic the behavior of multi-layer GCNs while avoiding graph over-smoothing. To leverage this observation, we develop a highly parallelizable random walk with restart engine to sample a fixed number of higher-order neighbors for each target node, which is stored in a database and can be efficiently prepared before model training.

In order to enable the contextualized embedding of MultiSage, the random walk engine needs to also retrieve the context nodes between each ego-neighbor pair of target nodes. However, it is not realistic to retrieve all intermediate context nodes if there are multiple, because it will blow up the size of the resulting contextualized neighbor list and pose challenges to the design of the contextualized graph convolution. It is also not necessary, because although each ego node can have multiple facets, usually only one facet is active when it interact with a particular neighbor node. Therefore, we build a novel light greedy algorithm based on stream data processing [235] into the random walk engine to retrieve only one dominant context node for each pair of target nodes, which best describe the interaction between them. The idea is to run many random walks between each pair of target nodes, and use the most frequently visited context node by these paths as the dominant context.

Hadoop2-based data provider. The second complexity lies in the retrieval and joining of node features. Consider a Pinterest graph with 1 billion pins, each with 64-dim *float* visual features and 128-dim *short-int* textual features. The *feature store* of all pins will thus take 1TB space. During training and inference, again since the neighbor of a node in the current batch can be anywhere on the graph, neighbor retrieval is fast enough only if the 1TB graph is completely stored in memory. This approach, though expensive, has been adopted for PinSage under the support of the Linux HugePage technology [228]. However, it is hard to acquire many dedicated machines with such large memory for rapid model development and training, and it constrains the inclusion of additional data and signals as a path to further model improvements.

To remove the requirements of large memory, we develop a Hadoop2-based data provider with pre-computed neighbor datasets and AWS S3 streaming. The idea is to sample and fix the neighbor lists $\mathcal{S}$ of all target nodes $\mathcal{T}$ on $\mathcal{G}$ with Algorithm 2 offline, pre-join the large feature stores with $\mathcal{S}$ and store the results on S3 cloud. During training and inference of MultiSage, it is then possible to dynamically prepare minibatches of nodes with neighbors, both already joined with the features, so as to avoid the heavy joining operation online. We develop an efficient and robust pipeline that handles tremendous amounts of data (more than 60TB intermediate data during a full join of all neighbor lists with the feature stores), and synchronize the model training and inference on multiple GPUs with S3 streaming through

dedicated stream data loaders.

5.3 EXPERIMENTAL EVALUATIONS

5.3.1 Experimental Setup

Dataset preparation. From Pinterest, we collect a pin-board graph of 76M pins, 15M boards, and 2.7B pin-board links. In addition, we collect a paper-venue graph of 87M papers, 46K venues and 87M paper-venue links from OAG [113] where all pairs of identical papers have different titles, to further evaluate the generalizability of MultiSage. Since in Pinterest and OAG, the most important use cases for search and recommendation are around pins and papers respectively, we model pins/papers as the target nodes, and boards/venues/keywords as the context nodes.

For Pinterest, we collect training data based on the *repin signals*. A repin pair $(q, i) \in \mathcal{R}$ is created when a user stores a pin i after exploring potentially related pins of q, which is a valuable signal indicating that the user likes the recommendation of i based on q (i and q are related in certain ways). A good model should be able to learn to capture the similarity between i and q. We remove repin pairs with low frequencies (*e.g.*, less than 2 times) to reduce noise, and keep a maximum number of repin pairs for the same query (*e.g.*, 20) to reduce bias. In our experiments, we collect a set of 75M repin pairs. We create separate validation sets from all training data by randomly sampling 1M from the 75M pairs.

For OAG, we collect 14M pairs of identical papers on Microsoft Academic Graph and Aminer. Since most pairs are easy to classify simply based on paper contents like titles, we further pick 72K from the 14M pairs with different titles for model training and evaluation. Similarly to Pinterest, a random set of 10K pairs are separated from the 72K pairs, which is only used for evaluation.

Compared algorithms. We mainly compare with the state-of-the-art production model PinSage currently deployed at Pinterest, together with a few other commonly used embedding methods leveraging various signals with advanced mainstream deep learning models [228]. The baselines we compare include

- **Visual**: The unified visual embedding used as pin features.
- **Textual**: The conceptnet textual embedding used as pin and paper features.
- **Combined**: The combination of visual and textual embeddings used as pin features.

- **Pixie** [112]: Graph-based ranking through aggregating random-walk visiting counts deployed at Pinterest, which essentially computes the popular Personalized PageRank scores [164].

- **PinSage** [228]: The state-of-the-art production pipeline that computes pin embeddings by incorporating visual, textual and graph signals based on the GraphSage model [33] currently deployed at Pinterest (V5).

- **GAT** [35]: Our implementation of graph attention networks based on PinSage V5 for scalable training.

- **HAN** [63]: Our implementation of heterogeneous graph attention networks based on PinSage V5 for scalable training.

Note that, only HAN and our MultiSage variants can model the context nodes between target nodes, whereas Pixie, PinSage and GAT can only model the homogeneous network of target nodes by ignoring their different interaction contexts.

Evaluation metrics. We evaluate the performance of different models based on a separate set of evaluation node pairs $\mathcal{P}_{eval}$. For each pair $p = (q, i) \in \mathcal{P}_{eval}$, we use q as a query and then compute a set of metrics based on the ranking position of i among the fixed pool $\mathcal{I}$ of evaluation nodes. The ranking is done based on exact cosine similarity for all embedding methods except *Pixie*, which directly returns a rank list of nodes given a query. Similar to [228], we compute the scaled Mean Reciprocal Rank as $MRR = \frac{1}{|\mathcal{P}_{eval}|} \sum_{(q,i)\in\mathcal{P}_{eval}} \frac{1}{R_{i,q}/M_s}$, where $R_{i,q}$ is the rank of i among $\mathcal{I}$. M_s is the scaling factor ensuring the difference between large ranks to be still noticeable. We use $M_s = 10$ which is smaller than 100 used in [228], because we now use a smaller pool $\mathcal{I}$ of nodes for faster evaluations and also because all baselines including PinSage have been largely improved in the past one year, which makes the distinguish between small ranks to be more important. Due to the same reasons, for *recall@K* (short as *REC@K*, also known as *hit-ratio@K*, which is defined as the fraction of queries q where i is ranked among the top K among $\mathcal{I}$), we also use extremely small values of K such as 10 and 1.

In addition to the ranking based metrics which only measure the relative distances among relevant and irrelevant nodes, we also compute the average cosine distances (DC) and Euclidean distances (DE) among the embeddings of all nodes ($DC*$ and $DE*$) and all relevant nodes pairs ($DC+$ and $DE+$). This helps us to understand the absolute distribution of nodes in the embedding space. A good embedding method should be able to spread all nodes further apart to occupy vast areas in the embedding space (large $D*$'s), while putting relevant nodes close to each other (small $D+$'s).

Furthermore, we compute the sets of top-K retrieved nodes for both q and i ($TopK(q)$ and $TopK(i)$, respectively), and thus compute the average sizes of intersection ($INT = |TopK(q) \cap TopK(i)|$) and union ($UNI = |TopK(q) \cup TopK(i)|$) of the two sets, as well as their *Jaccard index* ($JAC = INT/UNI$). These metrics help us to further understand how query nodes and positive nodes are distributed in the embedding space. A good embedding method should put q and i closer in the embedding space, in the sense that their neighborhoods $TopK(q)$ and $TopK(p)$ overlaps much, leading to large values of INT, small values of UNI, and large values of JAC.

Training Details For Pinterest, we collect training data mainly based on the *organic repin signals*. A repin pair $(q, i) \in \mathcal{R}$ is created when a user stores a pin i after exploring potentially related pins of q, which is a valuable signal indicating that the user likes the recommendation of i based on q (i and q are related in certain ways). A good model should be able to learn to capture the similarity between i and q. We remove repin pairs with low frequencies (*e.g.*, less than 2 times) to reduce noise, and keep a maximum number of repin pairs for the same query (*e.g.*, 20) to reduce bias. In our experiments, we collect a set of 75M repin pairs. We further create separate validation sets from all training data by randomly sampling 1M from the 75M pairs.

In addition to organic repins, we also extract pin pairs based on other signals to train and evaluate different models for comprehensive understanding of their behaviors. For example, we use *shopping signals* from Pinterest, where a shopping pair $(q, i) \in \mathcal{S}$ is created when a user *long-clicks* (click and stay for more than 10s) on a *shopping pin* (pins with links to shopping pages) i recommended for query pin q. The shopping signals, while also indicating successful recommendations, might favor different aspects of relatedness between i and q, which is helpful in testing the models' capability of capturing different contextual proximities among i and q.

We process the shopping pairs similarly as repin pairs to reduce noise and bias. However, in practice we observe shopping pairs to be much more noisy, and use human labors to further curate two categorized smaller sets of gold quality shopping pairs. Particularly, we pick out two popular categories of shopping products in Pinterest, *i.e.*, home decororation and women's fashion, and generate a set of 8192 gold pairs for each of them.

For OAG, we collect 14M pairs of identical papers on Microsoft Academic Graph and Aminer. Since most pairs are easy to classify simply based on paper contents like titles, we further pick 72K from the 14M pairs with different titles for model training and evaluation. Similarly to Pinterest, a random set of 10K pairs are separated from the 72K pairs, which is only used for evaluation.

As for model parameters, we use two separate three-layer feedforward neural networks

(FNN) with sizes $2349 \to 1024 \to 256$ for target embedding for Pinterest ($300 \to 256 \to 128$ for OAG), and use another two-layer FNN with sizes $600 \to 256$ for context embedding for Pinterest ($300 \to 128$ for OAG). A three-layer FNN with sizes $768 \to 128 \to 1$ is used for the computation of three-way attention weights for Pinterest ($384 \to 64 \to 1$ for OAG). All FNNs are with ReLU activations. On both datasets, we set the number of attention heads to 10, the training batch size to 256 and the number of epochs to 100K. We always use the Adam optimizer with learning rate 0.0001. The random walk parameters ζ, κ, ξ and s are empirically set to 10, 10K, 128, and 20, respectively, and the random walk is restarted with probability 0.1 at each target node. We deploy the data provider pipeline described in Section 5.2.3 on a Hadoop2 cluster with 378 d2.8xlarge Amazon AWS nodes. Model training is then done in parallel on a p2.16xlarge AWS machine with 8 GeForce GTX 1080 Ti GPUs.

5.3.2 Quantitative Evaluation

Overall comparison **Overall comparison.** We firstly compare MultiSage against all baseline methods on related pin recommendation based on the organic repin pairs, which is the most classic evaluation scenario in Pinterest due to its close connection to user engagement [228]. Similarly, we also present results on same paper identification based on the paper pairs on OAG. Table 5.2 shows the comprehensive set of evaluation metrics computed on both datasets. The differences in the performances between MultiSage and baselines all passed the standard paired t-test with p-value 0.01.

MultiSage-2 is our model on bipartite networks (*i.e.*, pin-board networks of Pinterest and paper-venue networks of OAG). As we can see, it constantly outperforms all baselines with significant margins in all cases, which provides strong signals towards its effectiveness and robustness in utilizing network contexts. In Particular, the performance gain over PinSage and GAT clearly demonstrates its broader model capacity regarding context nodes, while the improvements over HAN also corroborates its appropriate model design for handling the multipartite contexts.

To show that MultiSage is general and lends itself to model multipartite networks with more than two types of nodes, we further incorporate keywords into the OAG network to form a tripartite network by linking each paper to its keywords. During convolution (Algorithm 1), we compute two sets of context embeddings for each paper based on both the neighboring venues and keywords, which have the same neural architecture (Eq. 4-7) but different sets of learnable parameters. We observe that MultiSage-3, which is computed on the tripartite network, can lead to additional performance gain, thus indicating the generalizability of our

proposed system.

Pinterest	MRR	REC@1	REC@10	DC+	DC*	DE+	DE*	INT	UNI	JAC
Visual	0.4406	0.1710	0.3606	0.4194	0.6337	0.9101	1.1255	23.32	174.67	0.1506
Textual	0.5741	0.1888	0.4965	0.3414	0.7614	0.7549	1.2050	31.78	166.21	0.1917
Combined	0.4438	0.1731	0.3635	0.4190	0.6340	0.9096	1.1258	23.44	174.53	0.1512
Pixie	0.3093	0.0418	0.2169	N/A	N/A	N/A	N/A	21.32	176.65	0.1351
PinSage	0.8759	0.4928	0.8234	0.2655	0.9279	0.7161	1.3593	47.30	150.69	0.3302
GAT	0.8880	0.5357	0.8665	0.2532	0.9343	0.7060	1.3618	48.70	149.24	0.3572
HAN	0.9013	0.5653	0.8838	0.2501	0.9415	0.6907	1.3558	50.29	148.83	0.3672
MultiSage-2	**0.9569**	**0.6215**	**0.9326**	**0.2316**	**0.9655**	**0.6660**	**1.3871**	**53.95**	**144.04**	**0.3906**
OAG	MRR	REC@1	REC@10	DC+	DC*	DE+	DE*	INT	UNI	JAC
Textual	0.1418	0.0273	0.0399	0.1081	0.4788	0.2814	1.0557	33.10	164.87	0.2193
Pixie	0.3126	0.1054	0.2642	N/A	N/A	N/A	N/A	36.58	160.76	0.2517
PinSage	0.5682	0.1845	0.5193	0.1238	0.6381	0.3179	1.1577	41.13	156.80	0.2935
GAT	0.6059	0.2355	0.5498	0.1104	0.6416	0.2908	1.2022	43.02	155.70	0.3144
HAN	0.6214	0.2641	0.5749	0.1005	0.6543	0.2869	1.2383	44.96	154.49	0.3200
MultiSage-2	0.6874	0.3270	0.6455	0.0836	0.6989	**0.2542**	1.2769	48.63	148.97	0.3602
MultiSage-3	**0.7026**	**0.3614**	**0.6875**	**0.0814**	**0.7127**	0.2583	**1.3058**	**51.63**	**145.40**	**0.3891**

Table 5.2: Performance of state-of-the-art web-scale embedding methods for general recommendation.

Off-task analysis. In addition to related pin recommendation, we now focus on the comparison between MultiSage and the production model PinSage to evaluate how the learned pin embeddings can influence the performance of other important but not directly related tasks. For instance, besides the repin signals that quantify user engagement, shopping signals are impactful in monetization. However, since high-quality shopping pairs are expensive and scarce, can an embedding model trained based on repin signals be useful in shopping recommendation?

Method	Home Decoration			Women's Fashion		
	MRR	REC@1	REC@10	MRR	REC@1	REC@10
Visual	0.6446	0.2892	0.5615	0.5770	0.2568	0.4941
Textual	0.6066	0.3009	0.5420	0.4503	0.1881	0.3762
Combined	0.4438	0.1731	0.3635	0.5788	0.2576	0.4964
Pixie	0.2491	0.0344	0.1682	0.3394	0.0464	0.2576
PinSage	0.8021/0.8067	0.4195/0.4257	0.7439/0.7460	0.7537/0.7545	0.3754/0.3759	0.6838/0.6976
MultiSage	**0.8407/0.8488**	**0.4899/0.5160**	**0.7954/0.8146**	**0.8058/0.8294**	**0.4363/0.4711**	**0.7533/0.7806**

Table 5.3: Off-task utility of embeddings produced by different methods in shopping recommendation.

Table 5.3 presents the off-task utility of pin embeddings on shopping recommendation. As we can observe, the gap between MultiSage and PinSage change from those evaluated on repin pairs, but the advantage of MultiSage is clearly maintained, indicating its general beneficial effects on different related tasks. We further randomly take out half of the shopping pairs from the evaluation set and mix into the training data of repin pairs. Second values in

the last two rows of Table 5.3 are computed from embeddings trained with the mixture of repin and shopping pairs. As we can observe, MultiSage is able to improve significantly on the shopping metrics given additional shopping pairs for training, while the performance of PinSage almost stays the same. Note that, although related, similarity among repin pairs and shopping pairs might be slightly different. The results indicate that such subtle differences might be more effectively captured by MultiSage, which deliberately takes the contextualized proximity of nodes into account.

Ablation tests. To demonstrate the utility of our proposed techniques of contextual masking and contextual attention for contextualized multi-embedding, we compare multiple MultiSage variants with different model designs as introduced in the previous section, which are inspired by common practices in recent neural network models. We also conduct ablation tests with several variants of MultiSage:

- **trans**: Adding context, target embeddings before aggregation.
- **concat**: Concatenating context embeddings to target embeddings and computing a learnable projection before aggregation.
- **mask**: Applying the learnable contextual masking operation to the target embedding according to Eq. 5.5 before aggregation.
- **mask-diff**: Contextual mask plus difference based aggregation (more details in Section 3.3.1).
- **mask-atn (MultiSage)**: Contextual masking plus contextual attention based aggregation according to Eq. 5.7, which constitutes the final version of our proposed MultiSage model.

Figure 5.5 shows the main metrics we measured during the training of different model variants on the Pinterest graph. As we can observe, the full MultiSage model with contextual masking and contextual attention is able to converge most rapidly to stable performance and outperform all other model variants, indicating the rationality of our model design in achieving efficient and effective contextualized multi-embedding.

Scalability study. We study the scalability of our MultiSage pipelines from three perspectives: parameter complexity, runtime complexity, and storage complexity.

As we claim in Section 5.2, the model parameter complexity of MultiSage is independent of the graph sizes. Empirically, the training time complexity is linear with the number of training batches. In practice, as similar to PinSage, we find MultiSage to converge before a full epoch through all training pairs, which results in only a slightly longer runtime than PinSage within the same pipeline and hardware settings, due to the processing of additional training features and updating of additional model parameters. However, training on the

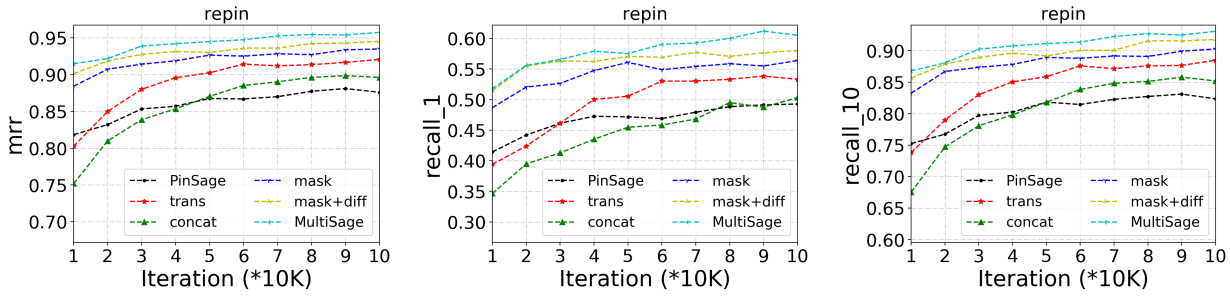


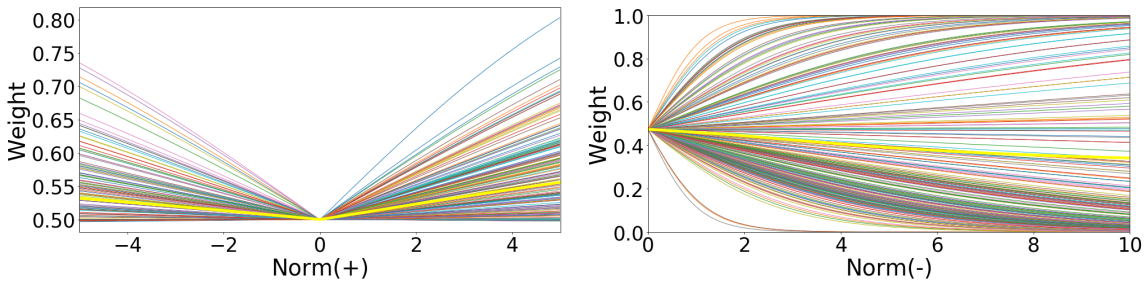
Figure 5.5: Performance of different model variants on *Pinterest*. Scores are computed every 10K iterations on the testing data.

AWS GPU machines synchronized with our Hadoop2-based data provider pipeline leads to around 25% reduce in runtime in comparison with the original Pinsage pipelines, while maintaining extremely close evaluation metrics.

Regarding storage, the original pipeline of PinSage requires the whole training graph (*e.g.*, over 2TB for the production graph in Pinterest) to be stored in memory for fast random access during neighbor aggregation, which was supported by Linux HugePage. As for our Hadoop2-based MultiSage pipeline, we totally remove this memory requirement by pre-joining and storing all data on AWS S3 and using dedicated stream data loaders to train the model on multiple GPUs. As a direct benefit, this new training pipeline enables much better accessibility of MultiSage model training, which allows more people to train the models on different machines simultaneously for different data and tasks. In the meantime, it enables the training with board features, and allows us to further increase the size of training graphs by adding various other features and signals upon availability.

5.3.3 Qualitative Exploration

Model visualizations. Although the usage of attention is intuitive for weighing the neighbors during graph convolution [35], it is intriguing but remains unknown how attention exactly weighs different neighbors. One interesting assumption could be that since GCN is ultimately conducting graph-based feature smoothing [210], attention might help stabilize this process by assigning less weight to more different neighbors based on the current embeddings. Based on this assumption, we designed the difference-based aggregation function by computing a weight for each neighbor $u \in \mathcal{N}_v$ based on the embedding distance distance between u and the ego v, which is shown to be beneficial in Figure 5.5 (mask+diff). Here, through particularly designed model visualizations, we aim to further study how the attention mechanism works and why it is better than simple difference-based weights.



(a) Summation of ego and neighbor

(b) Difference of ego and neighbor

Figure 5.6: Attention over different self-neighbor pairs.

IIn Figure 5.6, we fix the ego embedding as all zeros, while varying each of the 256 dimensions of neighbor embedding so as to change the norms of the (a) summation and (b) difference of ego and neighbor embeddings along the X-axis. Y-axis denotes the average attention weights yielded by the attention network learned on the Pinterest dataset. Each colored curve corresponds to the changes on one of the 256 dimensions, while the thick yellow curve denotes the average of all thin curves. As we can observe from both subfigures, attention learns to put different stress on different dimensions. Moreover, in (a), attention generally puts more weights to neighbors that sum up with egos to have larger norms, and in (b), it puts slightly less weights to neighbors that are more different from egos. In this sense, attention generalizes and is more powerful than simple difference-based weights.

Case studies. In the previous experiments, although we compute multi-embeddings during graph convolution, we only use the aggregated embeddings to evaluate the overall item similarity due to the lack of evaluation data for multi-embeddings. Now we showcase how to leverage the multi-embeddings we learn for fine-grained applications such as contextualized recommendation. Rather than generalized recommendation [228], in contextualized recommendation, we aim to rank items (*e.g.*, pins) based on contexts (*e.g.*, boards), which is flexibly personalized towards fine-grained semantics.

In Figure 5.7, we use two random real cases from Pinterest generated based on the learned MultiSage model to demonstrate its utility in the important but seldom studied task of contextualized recommendation in the real world. In particular, given a query pin of a fashion model (decorative accessory), rather than returning a list of generally relevant pins, we can choose arbitrary boards like fashion and crafts (Christmas and decoration), apply the corresponding learned board-based contextual masking on the base embedding of both query and candidates, and retrieve lists of pins that are relevant to the query in the corresponding perspectives. Such flexible contextualization can be easily combined with any existing search or recommendation services to enable fine-grained effective and interpretable personalization.

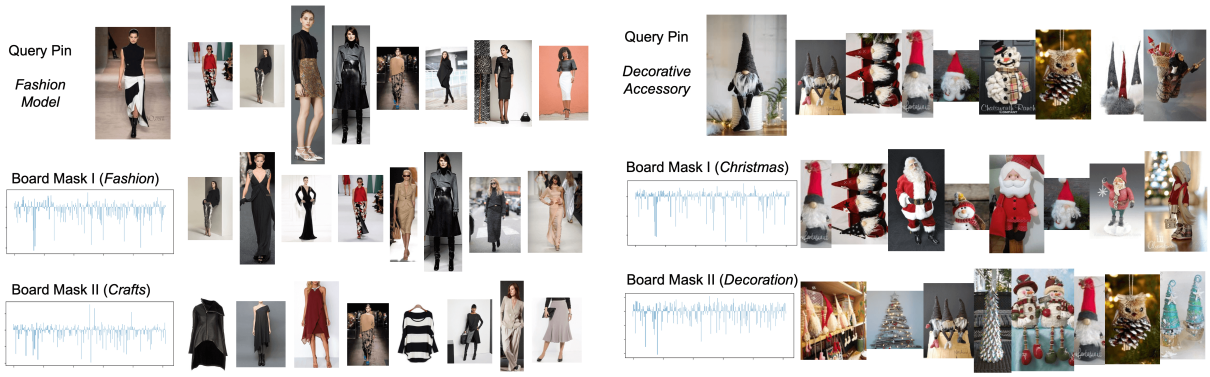


Figure 5.7: Examples of Pinterest pins recommended under different board-indicated semantic contexts.

5.4 NEW USER CHURN PREDICTION WITH IMPLICIT CONTEXT

Promoted by the widespread usage of internet and mobile devices, hundreds of online systems are being developed every year, ranging from general platforms like social media and e-commerce websites to vertical services including news, movie and place recommenders. As the market is overgrowing, the competition is severe too, with every platform striving to attract and keep more users.

While many of the world's best researchers and engineers are working on smarter advertisements to expand businesses by acquisition [236, 237], retention has received less attention, especially from the research community. The fact is, however, acquiring new users is often much more costly than retaining existing ones[2]. With the rapid evolution of mobile industry, the business need for better user retention is larger than ever before[3], for which, *accurate*, *scalable* and *interpretable* churn prediction plays a pivotal role[4].

Churn is defined as a user quitting the usage of a service. Existing studies around user churn generally take one of the two ways: data analysis and data-driven models. The former is usually done through user surveys, which can provide valuable insights into users' behaviors and mindsets. But the approaches require significant human efforts and are hard to scale, thus are not suitable for nowadays ubiquitous mobile apps. The development of large-scale data-driven models has largely improved the situation, but no existing work has looked into user behavior patterns to find the reasons behind user churn. As a consequence, the prediction results are less interpretable, and thus cannot fundamentally solve the problem of user churn.

[2]https://www.invespcro.com/blog/customer-acquisition-retention
[3]http://info.localytics.com/blog/mobile-apps-whats-a-good-retention-rate
[4]https://wsdm-cup-2018.kkbox.events

In this work, we take the anonymous data from Snapchat as an example to systematically study the problem of interpretable churn prediction. We notice that online platform users can be highly heterogeneous. For example, they may use (and leave) a social app for different reasons[5]. Through extensive data analysis on users' multi-dimensional temporal behaviors, we find it intuitive to capture this heterogeneity and assign users into different clusters, which can also indicate the various reasons behind their churn. Motivated by such observations, we develop ClusChurn, a framework that jointly models the types and churn of new users (Section 5.5.1).

To understand user types, we encounter the challenges of automatically discovering interpretable user clusters, addressing noises and outliers, and leveraging correlations among features. As a series of treatments, we apply careful feature engineering and adopt k-means with Silhouette analysis [238] into a three-step clustering mechanism. The results we get include six intuitive user types, each having typical patterns on both daily activities and ego-network structures. In addition, we also find these clustering results highly indicative of user churn and can be directly leveraged to generate type labels for users in an unsupervised manner (Section 5.5.2).

To enable interpretable churn prediction, we propose to jointly learn user types and user churn. Specifically, we design a novel deep learning framework based on LSTM [239] and attention [240]. Each LSTM is used to model users' temporal activities, and we parallelize multiple LSTMs through attention to focus on particular user types. Extensive experiments show that our joint learning framework delivers supreme performances compared with baselines on churn prediction with limited user activity data, while it also provides valuable insights into the reasons behind user churn, which can be leveraged to fundamentally improve retention (Section 5.5.3).

Note that, although we focus on the example of Snapchat data, our ClusChurn framework is general and able to be easily applied to any online platform with user behavior data. A prototype implementation of ClusChurn based on PyTorch is released on Github[6].

The main contributions of this work are summarized as follows:

1. Through real-world large-scale data analysis, we draw attention to the problem of interpretable churn prediction and propose to jointly model user types and churn.

2. We develop a general automatic new user clustering pipeline, which provides valuable insights into different user types.

[5]http://www.businessofapps.com/data/snapchat-statistics
[6]https://github.com/yangji9181/ClusChurn

3. Enabled by our clustering pipeline, we further develop a prediction pipeline to jointly predict user types and user churn and demonstrate its interpretability and supreme performance through extensive experiments.

4. We deploy ClusChurn as an analytical pipeline to deliver real-time data analysis and prediction to multiple relevant teams within Snap Inc. It is also general enough to be easily adopted by any online systems with user behavior data.

5.5 MODEL: CLUSTER-ENHANCED CHURN PREDICTION (CLUSCHURN)

5.5.1 Large-Scale Data Analysis

To motivate our study on user clustering and churn prediction, and gain insight into proper model design choices, we conduct an in-depth data analysis on a large real-world dataset from Snapchat. Sensitive numbers are masked for all data analysis within this paper.

Dataset We collect the anonymous behavior data of all new users who registered their accounts during the two weeks from August 1, 2017, to August 14, 2017, in a particular country. We choose this dataset because it is a relatively small and complete network, which facilitates our in-depth study on users' daily activities and interactions with the whole network. There are a total of 0.5M new users registered in the specific period, and we also collect the remaining about 40M existing users with a total of approximately 700M links in this country to form the whole network.

ID	Feat. Name	Feat. Description
0	chat_received	# textual messages received by the user
1	chat_sent	# textual messages sent by the user
2	snap_received	# snap messages received by the user
3	snap_sent	# snap messages sent by the user
4	story_viewed	# stories viewed by the user
5	discover_viewed	# discovers viewed by the user
6	lens_posted	# lenses posted to stories by the user
7	lens_sent	# lenses sent to others by the user
8	lens_saved	# lenses saved to devices by the user
9	lens_swiped	# lenses swiped in the app by the user

Table 5.4: Daily activities we collect for users on Snapchat.

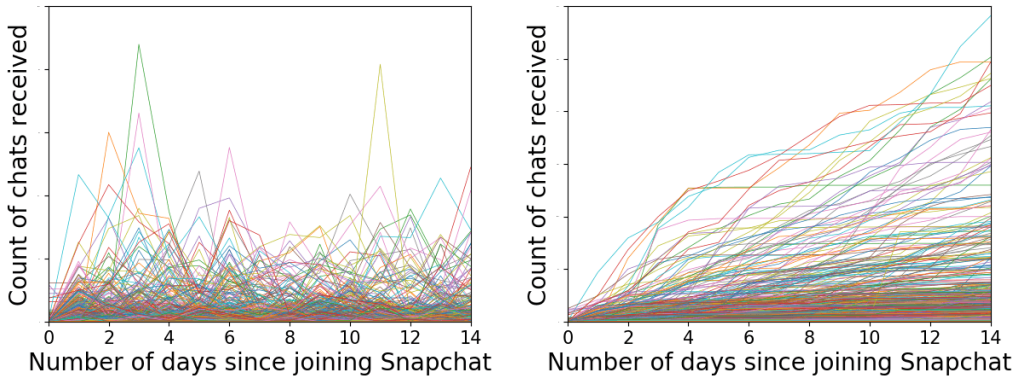
We leverage two types of features associated with users, *i.e.*, their daily activities and ego-network structures. Both types of data are collected during the two-week time window since each user's account registration. Table 5.4 provides the details of the daily activities data

we collect, which are from users' interactions with some of the core functions of Snapchat: *chat, snap, story, lens*. We also collect each user's complete ego-network, which are formed by her and her direct friends. The links in the networks are bi-directional friendships on the social app. For each user, we compute the following two network properties and use them as a description of her ego-network structures.

- Size: the number of nodes, which describes how many friends a user has.
- Density: the number of actual links divided by the number of all possible links in the network. It describes how densely a user's friends are connected.

As a summary, given a set of N users $\mathcal{U}$, for each user $u_i \in \mathcal{U}$, we collect her 10-dimensional daily activities plus 2-dimensional network properties, to form a total of 12-dimensional time series $\mathbf{A}_i$. The length of $\mathbf{A}_i$ is 14 since we collect each new user's behavioral data during the first two weeks after her account registration. Therefore, $\mathbf{A}_i$ is a matrix of 12×14.

Daily Activity Analysis Figure 5.8 (a) shows an example of daily measures on users' *chat_received* activities. Each curve corresponds to the number of chats received by one user every day during the first two weeks after her account registration. The curves are very noisy and bursty, which poses challenges to most time series models like HMM (Hidden Markov Models), as the critical information is hard to be automatically captured. Therefore we compute two parameters, *i.e.*, μ, the mean of daily measures to capture the activity volume, and l, the $lag(1)$ of daily measures to capture the activity burstiness. Both metrics are commonly used in time series analysis [241].



(a) Daily Measures (b) Aggregated Measures

Figure 5.8: Activities on *chat_received* in the first two weeks. Y-axis is masked in order not to show the absolute values.

Figure 5.8 (b) shows the aggregated measures on users' *chat_received* activities. Every curve corresponds to the total number of chats received by one user until each day after her

account registration. The curves have different steepness and inflection points. Motivated by a previous study on social network user behavior modeling [242], we fit a sigmoid function $y(t) = \frac{1}{1+e^{-q(t-\phi)}}$ to each curve, and use the two parameters q and ϕ to capture the shapes of the curves.

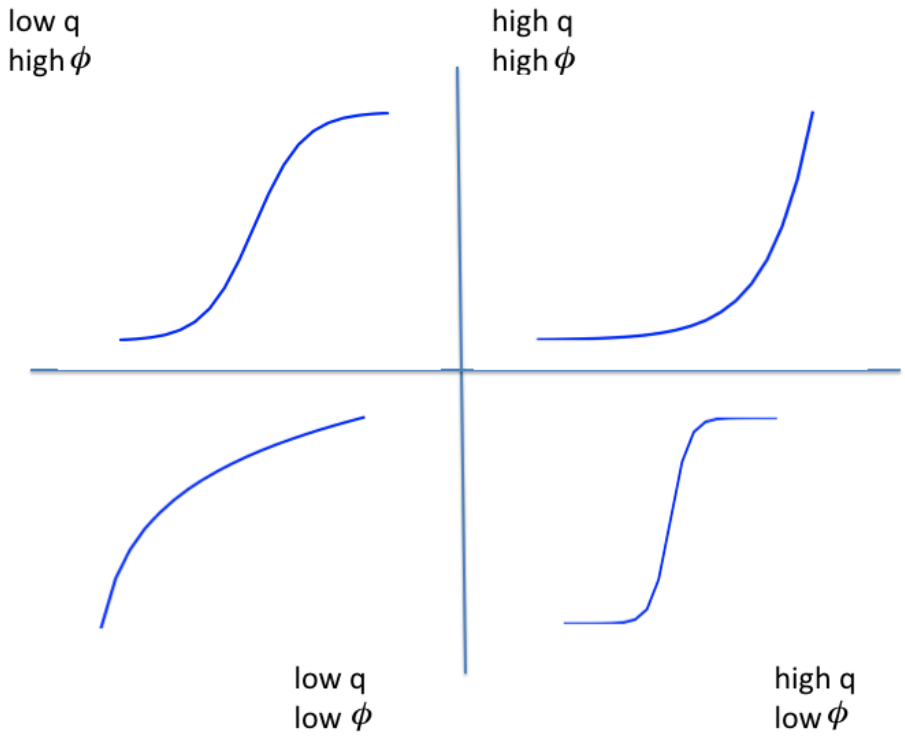


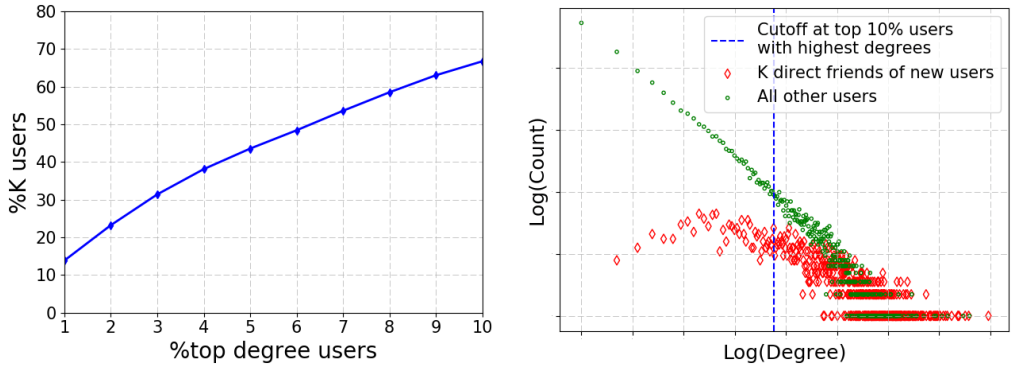
Figure 5.9: Main curve shapes captured by sigmoid functions with different parameter configurations.

Figure 5.9 shows 4 example shapes of curves captured by the sigmoid function with different q and ϕ values.After such feature engineering on the time series data, each of the 12 features is described by a vector of 4 parameters $\mathbf{f} = \{\mu, l, q, \phi\}$. We use $\mathbf{F}_i$ to denote the feature matrix of u_i and $\mathbf{F}_i$ is of size 12×4.

Network Structure Analysis In addition to daily activities, we also study how new users connect with other users. The 0.5M new users in our dataset directly make friends with a subset of a few million users in the whole network during the first two weeks since their account registration. We mask the absolute number of this group of users and use κ to denote it.

We find these κ users very interesting since there are about 114M links formed among them and 478M links to them. However, there are fewer than 700M links created in the

whole network of the total about 40M users in the country. It leads us to believe that there must be a small group of well-connected popular users in the network, which we call the *core* of a network, and in fact, this core overlaps with a lot of the κ direct friends of new users.



(a) Overlapping of core and the κ users (b) Degree distribution of the κ users

Figure 5.10: Most of the κ users are within the core.

To validate this assumption, we define the core of social networks as the set of users with the most friends, *i.e.*, nodes with highest degrees, motivated by earlier works on social network analysis [243]. Figure 5.10 (a) shows the percentage of the κ users within the core as we increase the size of the core from the top 1% nodes with highest degrees to the top 10%. Figure 5.10 (b) shows the particular degrees of the κ users drawn together with all other nodes, ordered by degrees on the x-axis. As we can see, 44% of the κ users are among the top 5% nodes with highest degrees, and 67% of them have 10% highest degrees. This result confirms our hypothesis that most links created by new users at the beginning of their journeys are around the network core. Since the κ direct friends do not entirely overlap with the core, it also motivates us to study how differently new users connect to the core, and what implications such differences can have on user clustering and churn prediction.

5.5.2 Interpretable User Clustering

In this section, we study what the typical new users are like on Snapchat and how they connect to the social network. We aim to find an interpretable clustering of new users based on their initial behaviors and evolution patterns when they interact with the various functions of a social app and other users. Moreover, we want to study the correlations between user types and user churn, so as to enable better churn prediction and personalized retention.

We also note that, besides churn prediction, interpretable user clustering is crucial for the

understanding of user behaviors so as to enable various product designs, which can ultimately lead to different actions towards the essentially different types of users. Therefore, while we focus on the end task of churn prediction, the framework proposed in this work is generally useful for any downstream applications that can potentially benefit from the understanding of user types, such as user engagement promotion.

Challenges Automatically finding interpretable clustering of users *w.r.t.*multi-dimensional time series data poses quite a few challenges, which makes the canonical algorithms for clustering or feature selection such as k-means and principal component analysis impractical [244].

Challenge 1: Zero-shot discovery of typical user types. As we discuss in Section 5.4, users are often heterogeneous. For example, some users might actively share contents, whereas others only passively consume [245]; some users are social hubs that connect to many friends, while others tend to keep their networks neat and small [246]. However, for any arbitrary social app, is there a general and systematic framework, through which we can automatically discover the user types, without any prior knowledge about possible user types or even the proper number of clusters?

Challenge 2: Handling correlated multi-dimensional behavior data. Users interact with a social app in multiple ways, usually by accessing different functions of the app as well as interacting with other users. Some activities are intuitively highly correlated, such as *chat_sent* and *chat_received*, whereas some correlations are less obvious, such as *story_viewed* and *lens_sent*. Moreover, even highly correlated activities cannot be simply regarded as the same. For example, users with more chats sent than received are quite different from users in the opposite. Therefore, what is a good way to identify and leverage the correlations among multiple dimensions of behavior data, including both functional and social activities?

Challenge 3: Addressing noises and outliers. User behavior data are always noisy with random activities. An active user might pause accessing the app for various hidden reasons, and a random event might cause a dormant user to be active for a period of time as well. Moreover, there are always outliers, with extremely high activities or random behavior patterns. A good clustering framework needs to be robust to various kinds of noises and outliers.

Challenge 4: Producing interpretable clustering results. A good clustering result is useless unless the clusters are easily interpretable. In our scenario, we want the clustering framework to provide insight into user types, which can be readily turned into actionable items to facilitate downstream applications such as fast-response and targeted user retention.

Methods To deal with those challenges, we design a robust three-step clustering framework. Consider a total of two features, namely, $\mathbf{f}^1$ (*chat_received*) and $\mathbf{f}^2$ (*chat_sent*), for four users, u_1, u_2, u_3 and u_4. Figure 5.11 illustrates a toy example of our clustering process with the details described in the following.

Step 1: Single-feature clustering. For each feature, we apply k-means with Silhouette analysis [238] to automatically decide the proper number of clusters K and assign data into different clusters. For example, as illustrated in Figure 5.11, for *chat_received*, we have the feature of four users $\{\mathbf{f}_1^1, \mathbf{f}_2^1, \mathbf{f}_3^1, \mathbf{f}_4^1\}$, each of which is a 4-dimensional vector (*i.e.*, $\mathbf{f} = \{u, l, q, \phi\}$). Assume K chosen by the algorithm is 3. Then we record the cluster belongingness, *e.g.*, $\{l_1^1 = 1, l_2^1 = 1, l_3^1 = 2, l_4^1 = 3\}$, and cluster centers $\{\mathbf{c}_1^1, \mathbf{c}_2^1, \mathbf{c}_3^1\}$. Let's also assume that for *chat_sent*, we have $K = 2$, $(l_1^2 = 1, l_2^2 = 1, l_3^2 = 1, l_4^2 = 2)$ and $\{\mathbf{c}_1^2, \mathbf{c}_2^2\}$. This process helps us to find meaningful types of users *w.r.t.*every single feature, such as users having high volumes of *chat_received* all the time versus users having growing volumes of this same activity day by day.

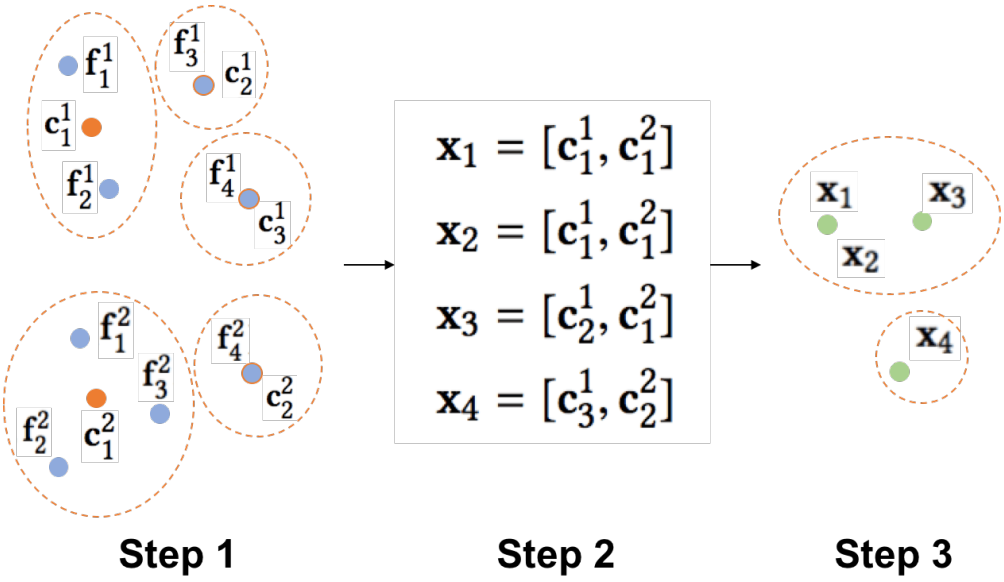


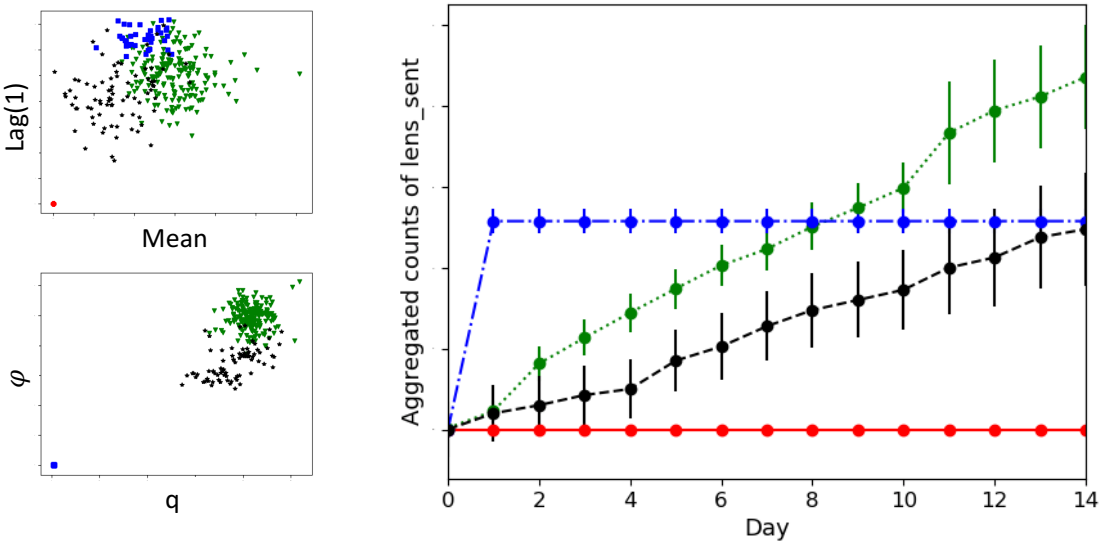
Figure 5.11: A toy example of our 3-step clustering framework.

Step 2: Feature combination. We convert the features of each user into a combination of the features of her nearest cluster center in each feature. Continue our toy example in Figure 5.11. Since user u_1 belongs to the first cluster in feature *chat_received* and first cluster in feature *chat_sent*, it is replaced by $\mathbf{x}_1$, which is a concatenation of $\mathbf{c}_1^1$ and $\mathbf{c}_1^2$. u_2, u_3 and u_4 are treated in the same way. This process helps us to largely reduce the influence of noises and outliers because every single feature is replaced by that of a cluster center.

Step 3: Multi-feature clustering. We apply k-means with Silhouette analysis again on the feature combinations. As for the example, the clustering is done on $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$. The algorithm explores all existing combinations of single-dimensional cluster centers, which record the typical values of combined features. Therefore, the multi-feature clustering results are the typical combinations of single-dimensional clusters, which are inherently interpretable.

Results

Clustering on single features. We first present our single-feature clustering results on each of users' 12-dimensional behaviors. Figure 5.12 provides the detailed results on *lens_sent* as an example. The results on other features are slightly different regarding the numbers of clusters, shapes of the curves, and numbers of users in each cluster. However, the method used is the same and they are omitted to keep the presentation concise.



(a) Parameter dist. (b) Activity patterns.

Figure 5.12: 4 types of users shown with different colors.

Figure 5.12 (a) shows the four parameters we compute over the 14-day period on users' *lens_sent* activities, as they distribute into the four clusters detected by the k-means algorithm. The number of clusters is automatically selected with the largest average Silhouette score when k is iterated from 2 to 6, which corresponds to clusters that are relatively far away from each other while having similar sizes. Figure 5.12 (b) shows the corresponding four types of users with different activity patterns on *lens_sent*. The first type of users (red) have no activity at all, while the second type (green) have stable activities during the two weeks.

Type 3 users (blue) are only active in the beginning, and type 4 users (black) are occasionally active. These activity patterns are indeed well captured by the volume and burstiness of their daily measures, as well as the shape of the curves of their aggregated measures. Therefore, the clusters are highly interpretable. By looking at the clustered curves, we can easily understand the activity patterns of each type of users.

Clustering on network properties. For single-feature clustering on network properties, as we get four clusters on ego-network size and three clusters on density, there is a total of $4 \times 3 = 12$ possible combinations of different patterns. However, when putting these two features of network properties together with the ten features of daily activities through our multi-feature clustering framework, we find that our new users only form three typical types of ego-networks. This result proves the efficacy of our algorithm since it automatically finds that only three out of the twelve combinations are essentially typical.

Figure 5.13 illustrates three example structures. The ego-networks of type 1 users have relatively large sizes and high densities; type 2 users have relatively small ego-network sizes and low densities; users of type 3 have minimal values on both measures.

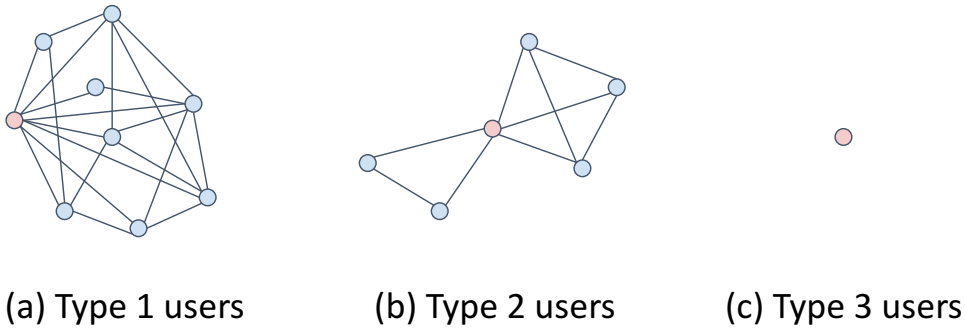


Figure 5.13: Examples of 3 types of ego-network structures.

Through further analysis, we find that these three types of new users clustered by our algorithm based on the features of their ego-networks have strong correlations with their positions in the whole social network. Precisely, if we define network core as the top 5% users that have the most friends in the entire network, and depict the whole network into a jellyfish structure as shown in Figure 5.14, we can exactly pinpoint each of the three user types into the tendrils, outsiders, and disconnected parts. Specifically, type 1 users are mostly tendrils with about 58% of direct friends in the core; type 2 users are primarily outsiders with about 20% of direct friends in the core; type 3 users are mostly disconnected with almost no friends in the core. Such result again proves that our clustering framework can efficiently find important user types.

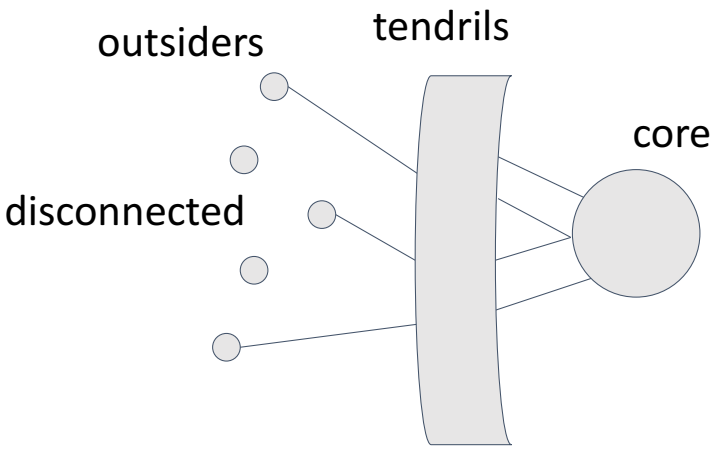


Figure 5.14: The whole network depicted into a jellyfish shape.

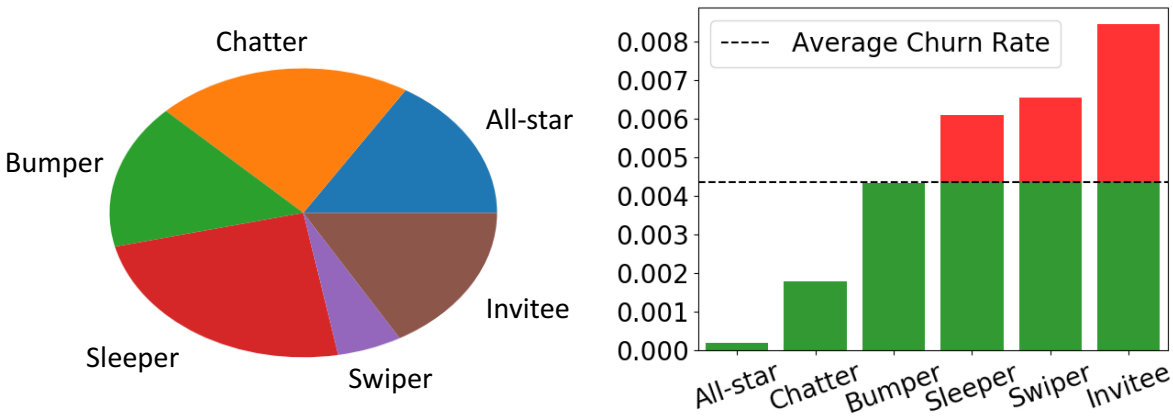
Clustering on all behaviors. Combining new users' network properties with their daily activities, we finally come up with six cohorts of user types, which is also automatically discovered by our algorithm without any prior knowledge. Looking into the user clusters, we find their different combinations of features quite meaningful, regarding both users' daily activities and ego-network structures. Subsequently, we are able to give the user types intuitive names, which are shown in Table 5.5. Figure 5.15 (a) shows the portions of the six types of new users.

We define a user churns if there is no activity at all in the second week after account registration. To get more insight from the user clustering results and motivate an efficient churn prediction model, we also analyze the churn rate of each type of users and present the results in Figure 5.15 (b). The results are also very intuitive. For example, All-star users are very unlikely to churn, while Swipers and Invitees are the most likely to churn.

ID	Type Name	Daily Activities	Ego-network Type
0	All-star	Stable active chat, snap, story & lens	Tendril
1	Chatter	Stable active chat & snap, few other acts	Tendril
2	Bumper	Unstable chat & snap, few other acts	Tendril
3	Sleeper	Inactive	Disconnected
4	Swiper	Active lens swipe, few other acts	Disconnected
5	Invitee	Inactive	Outsider

Table 5.5: 6 types of new users and their characteristics.

Note that, our new user clustering results are highly intuitive, and in the meantime provide a lot of valuable insights. For example, the main differences between All-star users and Chatters are their activities on *story* and *lens*, which are the additional functions of Snapchat. Being active in using these functions indicates a much lower churn rate. The small group of Swipers is impressive too since they seem to only try out the lenses a lot without utilizing



(a) Portions

(b) Churn rates

Figure 5.15: Portions and churn rates of the six new user types. The y-axis is rescaled to not show the absolute values.

any other functions of the app, which is related to an entirely high churn rate. Quite a lot of new users seem to be invited to the app by their friends, but they are highly likely to quit if not interacting with their friends, exploring the app functions or connecting to core users. Insights like these are highly valuable for user modeling, growth, retention and so on.

Although we focus our study on Snapchat data in this paper, the clustering pipeline we develop is general and can be applied to any online platforms with multi-dimensional user behavior data. The code of this pipeline has also been made publicly available.

5.5.3 Fast-Response Churn Prediction

Motivated by our user type analysis and the correlations between user types and churn, we aim to develop an efficient algorithm for interpretable new user churn prediction. Our analysis of real data shows that new users are most likely to churn in the very beginning of their journey, which urges us to develop an algorithm for fast-response churn prediction. The goal is to accurately predict the likelihood of churn by looking at users' very initial behaviors, while also providing insight into possible reasons behind their churn.

Challenges New user churn prediction with high accuracy and limited data is challenging mainly for the following three reasons.

Challenge 1: Modeling sequential behavior data. As we discuss in Section 5.5.1.1, we model each new user by their initial interactions with different functions of the social app as well as their friends, and we collect a 12-dimensional time series $\mathbf{A}_i$ for each new user $u_i \in \mathcal{U}$.

However, unlike for user clustering where we leverage the full two-week behavior data of each user, for fast-response churn prediction, we only focus on users' very limited behavior data, *i.e.*, from the initial few days. The data are naturally sequential with temporal dependencies and variable lengths. Moreover, the data are very noisy and bursty. These characteristics pose great challenges to traditional time series models like HMM.

Challenge 2: Handling sparse, skewed and correlated activities. The time series activity data generated by each new user are multi-dimensional. As we show in Section 5.5.2, such activity data are very sparse. For example, *Chatters* are usually only active in the first four dimensions as described in Table 5.4, while *Sleepers* and *Invitees* are inactive in most dimensions. Even *All-star* users have a lot of 0's in certain dimensions. Besides the many 0's, the distributions of activity counts are highly skewed instead of uniform and many activities are correlated, like we discuss in Section 5.5.2.1.

Challenge 3: Leveraging underlying user types. As shown in our new user clustering analysis and highlighted in Figure 5.15 (b), our clustering of new users is highly indicative of user churn and should be leveraged for better churn prediction. However, as we only get access to initial several days instead of the whole two-week behaviors, user types are also unknown and should be jointly inferred with user churn. Therefore, how to design the proper model that can simultaneously learn the patterns for predicting user types and user churn poses a unique technical challenge that cannot be solved by existing approaches.

Methods and Results We propose a series of solutions to treat the challenges listed above. Together they form our efficient churn prediction framework. We also present comprehensive experimental evaluations for each proposed model component. Our experiments are done on an anonymous internal dataset of Snapchat, which includes 37M users and 697M bi-directional links. The metrics we compute include accuracy, precision, and recall, which are commonly used for churn prediction and multi-class classification [247]. The baselines we compare with are logistic regression and random forest, which are the standard and most widely practiced models for churn prediction and classification. We randomly split the new user data into training and testing sets with the ratio 8:2 for 10 times, and run all compared algorithms on the same splits to take the average performance for evaluation. All experiments are run on a single machine with a 12-core 2.2GHz CPU and no GPU, although the runtimes of our neural network models can be largely improved on GPUs.

Solution 1: Sequence-to-sequence learning with LSTM. The intrinsic problem of user behavior understanding is sequence modeling. The goal is to convert sequences of arbitrary lengths with temporal dependences into a fixed-length vector for further usage. To this

end, we propose to leverage the state-of-the-art sequence-to-sequence model, that is, LSTM (Long-Short Term Memory) from the family of RNN (Recurrent Neural Networks) [239, 117]. Specifically, we apply a standard multi-layer LSTM to the multi-dimensional input $\mathbf{A}$. Each layer of the LSTM computes the following functions

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ c_t &= f_t * c_{t-1} + i_t * \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(c_t) \end{aligned} \tag{5.8}$$

where t is the time step in terms of days, h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the hidden state of the previous layer at time t, with $x_t = a_{.t}$ for the first layer, and i_t, f_t, o_t are the input, forget and out gates, respectively. σ is the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$. Dropout is also applied to avoid overfitting. We use Θ_l to denote the set of parameters in all LSTM layers.

A linear projection with a sigmoid function is connected to the output of the last LSTM layer to produce user churn prediction as

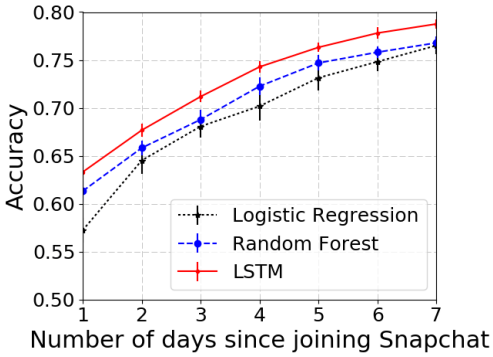
$$\hat{y} = \sigma(\mathbf{W}_c o_T + \mathbf{b}_c). \tag{5.9}$$

We use Θ_c to denote the parameters in this layer, *i.e.*, $\mathbf{W}_c$ and $\mathbf{b}_c$.

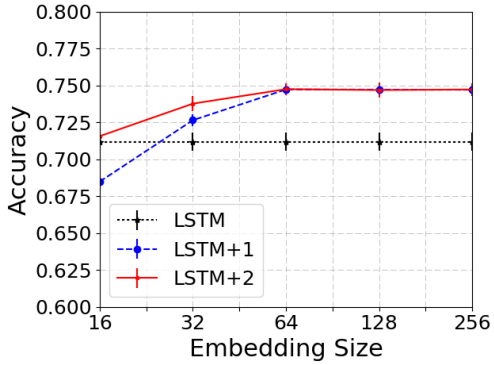
Unlike standard methods for churn prediction such as logistic regression or random forest, LSTM is able to model user behavior data as time series and capture the evolvement of user activities through recognizing the intrinsic temporal dependencies. Furthermore, compared with standard time series models like HMM, LSTM is good at capturing both long term and short term dependences within sequences of variable lengths. When the lengths are short, LSTM acts similarly as basic RNN [117], but when more user behaviors become available, LSTM is expected to excel.

Figure 5.16 (a) shows the performances of compared models. The length of the output sequence of LSTM is empirically set to 64. In the experiments, we vary the amounts of user behavior data the models get access to and find that more days of behavior data generally lead to better prediction accuracy. We can also see that new users' initial activities in the first few days are more significant in improving the overall accuracy. A simple LSTM model can outperform all compared baselines with a substantial margin. The runtime of LSTM on CPU is within ten times of the runtimes of other baselines, and it can be significantly

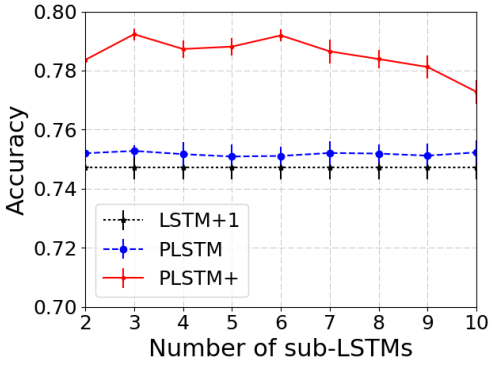
improved on GPUs.



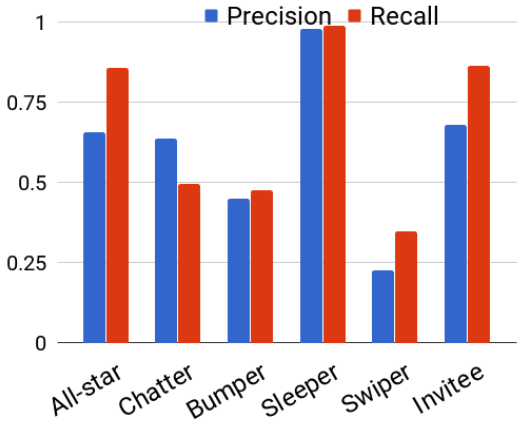
(a) Single LSTM



(b) Activity embedding



(c) Parallel LSTMs



(d) User type prediction

Figure 5.16: Comprehensive experimental results on our churn prediction framework compared with various baseline methods.

Solution 2: LSTM with activity embedding. To deal with sparse, skewed and correlated activity data, we propose to add an activity embedding layer in front of the standard LSTM layer. Specifically, we connect a fully connected feedforward neural network to the original daily activity vectors, which converts users' sparse activity features of each day into distributional activity embeddings, while deeply exploring the skewness and correlations of multiple features through the linear combinations and non-linear transformations. Specifically, we have

$$e_{\cdot t} = \psi^H(\dots\psi^2(\psi^1(a_{\cdot t}))\dots), \tag{5.10}$$

where

$$\psi^h(e) = \text{ReLU}(W_e^h \text{Dropout}(e) + b_e^h). \tag{5.11}$$

H is the number of hidden layers in the activity embedding network. Θ_e is the set of parameters in these H layers. With the activity embedding layers, we simply replace $\mathbf{A}$ by $\mathbf{E}$ for the input of the first LSTM layer, with the rest of the architectures unchanged.

Figure 5.16 (b) shows the performances of LSTM with activity embedding of varying number of embedding layers and embedding sizes. The length of the output sequence of LSTM is kept as 64. The overall performances are significantly improved with one single layer of fully connected non-linear embedding (*LSTM+1*), while more layers (*e.g.*, *LSTM+2*) and larger embedding sizes tend to yield similar performances. The results are intuitive because a single embedding layer is usually sufficient to deal with the sparsity, skewness, and correlations of daily activity data. We do not observe significant model overfitting due to the dropout technique and the large size of our data compared with the number of model parameters.

Solution 3: Parallel LSTMs with joint training. To further improve our churn prediction, we pay attention to the underlying new user types. The idea is that, for users in the training set, we get their two-week behavior data, so besides computing their churn labels y based on their second-week activities, we can also compute their user types $\mathbf{t}$ with our clustering framework. For users in the testing set, we can then compare the initial behaviors with those in the training set to guess their user types, and leverage the correlation between user types and churn for better churn prediction.

To implement this idea, we propose parallel LSTMs with joint training. Specifically, we assume there are K user types. K can be either chosen automatically by our clustering framework or set to specific values. Then we jointly train K sub-LSTMs on the training set. Each sub-LSTM is good at modeling one type of users. We parallelize the K sub-LSTMs and merge them through attention [240] to jointly infer hidden user types and user churn.

As shown in Figure 5.17, for each user, the input of a sequence of activity embedding vectors $\mathbf{E}$ is put into K sub-LSTMs in parallel to generate K *typed sequences*:

$$\mathbf{s}_k = \text{LSTM}_k(\mathbf{E}). \tag{5.12}$$

To differentiate hidden user types and leverage this knowledge to improve churn prediction, we introduce an attention mechanism to generate user behavior embeddings by focusing on their latent types. A positive attention weight w_k is placed on each user type to indicate

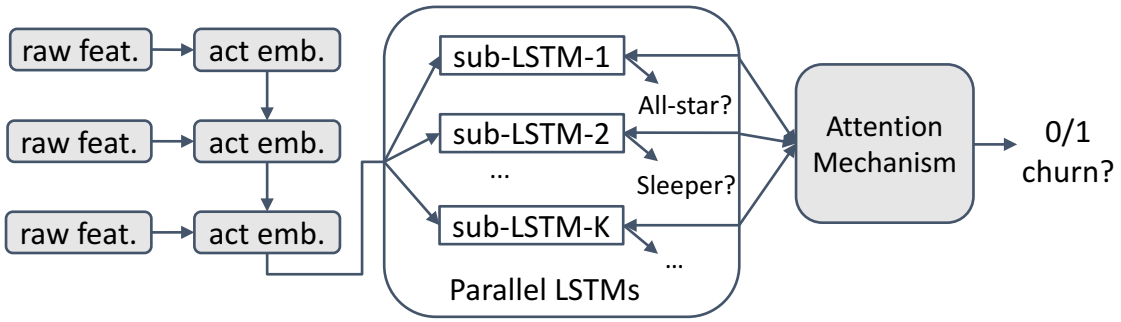


Figure 5.17: Parallel LSTMs with user type attention.

the probability of the user to be of a particular type. We compute w_k as a similarity of the corresponding typed sequence $\mathbf{s}_k$ and a global unique *typing vector* $\mathbf{v}_t$, which is jointly learned during the training process.

$$w_k = \text{softmax}(\mathbf{v}_t^T \mathbf{s}_k). \tag{5.13}$$

Here softmax is taken to normalize the weights and is defined as $\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$. The user behavior embedding $\mathbf{u}$ is then computed as a sum of the typed sequences weighted by their importance weights:

$$\mathbf{u} = \sum_{k=1}^{K} w_k \mathbf{s}_k. \tag{5.14}$$

The same linear projection with sigmoid function as in Eq. 5.9 is connected to $\mathbf{u}$ to predict user churn as binary classification.

$$\hat{y} = \sigma(\mathbf{W}_c \mathbf{u} + \mathbf{b}_c). \tag{5.15}$$

To leverage user types for churn prediction, we jointly train a typing loss l_t and a churn loss l_c. For l_t, we firstly compute users' soft clustering labels $\mathbf{Q}$ as

$$q_{ik} = \frac{(1 + ||\mathbf{f}_i - \mathbf{c}_k||^2)^{-1}}{\sum_j (1 + ||\mathbf{f}_i - \mathbf{c}_j||^2)^{-1}}. \tag{5.16}$$

q_{ik} is a kernel function that measures the similarity between the feature $\mathbf{f}_i$ of user u_i and the cluster center $\mathbf{c}_k$. It is computed as the probability of assigning u_i to the kth type, under the assumption of Student's t-distribution with degree of freedom set to 1 [248].

We use w_{ik} to denote the attention weight for user u_i on type t_k. Thus, for each user u_i,

we compute her typing loss as the cross entropy on $q_{i\cdot}$ and $w_{i\cdot}$. So we have

$$l_t = -\sum_i \sum_k q_{ik} \log(w_{ik}). \tag{5.17}$$

For l_c, we simply compute the log loss for binary predictions as

$$l_c = \sum -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i), \tag{5.18}$$

where y_i is the binary ground-truth churn label and $\hat{y}_i$ is the predicted churn label for user u_i, respectively.

Subsequently, the overall objective function of our parallel LSTM with joint training is

$$l = l_c + \lambda l_t, \tag{5.19}$$

where λ is a hyper-parameter controlling the trade-off between churn prediction and type prediction. We empirically set it to a small value like 0.1 in our experiments.

Figure 5.16 (c) shows the performances of parallel LSTMs with and without joint training (*PLSTM+* vs. *PLSTM*). The only difference between the two frameworks is that *PLSTM* is not trained with the correct user types produced by our clustering framework. In the experiments, we vary the number of clusters and sub-LSTMs and find that joint training is always significantly helpful. The performance of parallel LSTMs with joint training peaks with 3 or 6 sub-LSTMs. While the number 3 may accidentally align with some trivial clusters, the number 6 actually aligns with the six interpretable cohorts automatically chosen by our clustering framework, which illustrates the coherence of our two frameworks and further supports the sanity of splitting the new users into six types.

Besides churn prediction, Figure 5.16 (d) shows that we can also predict what type a new user is by looking at her initial behaviors rather than two-week data, with different precisions and recalls. Our algorithm is good at capturing *All-star*, *Sleeper* and *Invitee*, due to their distinct behavior patterns. *Swiper* and *Bumper* are harder to predict because their activity patterns are less regular. Nonetheless, such fast-response churn predictions with insights into user types can directly enable many actionable production decisions such as fast retention and targeted promotion.

CHAPTER 6: CONCLUSIONS

In my doctoral research, I systematically studied the organization, modeling, and application of multi-facet context-rich graphs under the novel framework of contextualized projections. The future of graph data mining lies in the principled integration of graph data with various other types of data, including temporal, spatial, textual, visual contents and much beyond. Guided by this vision, my long-term research goal is to foster a wholesome ecosystem of multi-facet graphs, where appropriate semantic contexts can be recognized according to specific graph mining tasks to build up the backbone of contextualized projections, upon which graph data can be generated and stored with the corresponding topology and context, and subsequent task-specific learning and inference can be conducted in a principled and efficient paradigm. Below are some specific future directions worth exploring as the extension beyond my past research on multi-facet graph mining with contextualized projections.

Multi-disciplinary open knowledge extraction and exploitation. By leveraging the interconnections among data, network models are distinct and complementary to all models over unary data. Existing practices are mainly around semi-supervised learning based on the homophily and data manifold assumptions, with graph smoothness regularizations for traditional tasks like the ranking and classification of images and documents. However, these tasks are often single-disciplinary, that is, they do not sufficiently explore and model the interactions among multi-disciplinary contexts and structures in a joint way. For example, a well-trained computer vision model can identify the image of a store, and the leverage of an image similarity graph can help reduce the required amount of labeled training data. However, through the integration of models from other disciplines, surrounding attributes like opening times and texts like place names can be used to further infer the category of the store (*e.g.*, retail, restaurant, bar). Moreover, graphs derived from multiple disciplines like social networks and sensor networks can be further leveraged to infer the popularity of and traffic conditions around the place. Most importantly, the context and graph models can be combined to collectively infer the different properties of interconnected objects in closed loops to harvest the mutual enhancement among different disciplines (*e.g.*, spatial-temporal signals from physical GPS sensors in smartphones can aid the entity disambiguation and event detection within textual data from social platforms, whereas entities and events automatically detected from texts like reviews and tweets can in turn provide meaningful details about the places), which I believe is the key to bridge the gap between domain-specific and general AI. My future research on multi-facet graph mining will exactly address the challenges in the *principled fusion of multi-disciplinary models for graph-based collective*

learning, via techniques like multi-facet contextualized data completion, structured reasoning, and representation learning.

Context-aware deeper insights into network dynamics. Traditional theoretical analysis and modeling of graph data are mainly focused on raw graphs (*i.e.*, graphs without contexts like node attributes and link types). Nonetheless, the studies are fruitful regarding the insights into network dynamics underlying interconnected objects (*e.g.*, spectral analysis, stochastic block models, invariance theory). Due to the surge of deep learning, graph theoretical studies are brought back for the deeper understanding and interpretable improvements of popular neural network based graph embedding models like DeepWalk and GCN. However, the studies are still around raw graphs and less useful for graphs with rich contexts. While it is intuitive that nodes with different contexts can interact in rather different patterns (*e.g.*, social leaders and cancer cells often form dense cores centered around them while new users and benign cells often only weakly connect to those cores), little theoretical effort has been made to understand the indications between network context and topology, to answer essential and important questions like: *Do node attributes help network embedding?* (seemingly straightforward, the answer might be counterintuitive due to insights from graph invariance theory and recent empirical observations that both show random node features to be competitive to real node attributes in certain scenarios); *When do GCNs excel (or collapse) on text-rich networks?* (deeper modeling into non-categorical node attributes like texts might uncover the weaknesses of current GCN architectures, particularly the uniform linear aggregation of direct neighbors, and thus hint on novel context-driven GCN design principles); *Can structures and contexts be aligned?* (Assuming both structures and contexts of networks are generated by a unique underlying mechanism, powerful nonlinear neural embedding models should allow us to compute the high-level shared network representations across structures and contexts, from where contexts can be translated to structures and vice versa, which leads to crystal clear understanding towards network dynamics). Driven by these puzzles, my future research on multi-facet graph mining will thus also aim at *combining theoretical insights and empirical observations for interpretable network modeling in the context-rich setting, with the end goal of establishing a principled mapping between complex structures and multi-facet contexts on networks.*

Cube-empowered large-scale interactive graph mining. In many applications, acquiring knowledge from data is an interactive process where people and machines need to collaborate with each other. There is great potential in leveraging my CubeNet system to facilitate such a human-in-the-loop process: (1) machines accept user-selected data, perform

network analysis along different facets and granularities, and provide summarized knowledge with visualizations (*e.g.*, contrastive network pattern mining); and (2) users make sense of the resultant patterns and visual clues, adjust their data selection schemas, and provide feedbacks to guide the machines to extract more useful knowledge. Along this line, my future research on multi-facet graph mining will address challenges such as the design of large-scale cube materialization strategies and systems that return user-desired results in real or near-real time, cube-tailored user-friendly visualization techniques and interaction interfaces, as well as effective policy learning for intelligent cube exploration upon user feedbacks.

Graph model efficiency and privacy in distributed settings. While the deep integration of multi-facet contexts from different data sources through collective learning in the network setting is powerful and intriguing, the unrestricted modeling and exchanging of multi-disciplinary data may be prone to training efficiency bottlenecks and collective attacks that seriously threaten systems' robustness and users' privacy. For example, structural information such as specific graph patterns shall be leveraged to enable specific information exchange channels, but it can also compromise data privacy through the comparison between auxiliary graphs and anonymized graphs (*i.e.*, membership inference). Even without the access to actual data, since structural information is largely captured by nowadays powerful neural network models like generative adversarial networks, properly designed adversarial attackers can still accomplish model inversion and leakage attacks against collaborative deep learning. Worse still, the leakage of privacy data such as user identities and user-item interactions on one domain (*e.g.*, social networks) can easily propagate to other collectively modeled domains (*e.g.*, e-commerce, cyber-physical and even financial), which is then extremely dangerous and concerning. To prevent such attacks from happening, my future work on multi-facet graph mining will also strive to guarantee data privacy along the development of effective (intimidating) data mining methods. Particularly, I plan to achieve this through two powerful security principles, *i.e.*, differential privacy (DP) and secure multi-party computation (MPC), both of which have hardly been studied in the network data mining scenario. By enforcing DP on single-source graph models, I target on the optimal trade-off between model utility and data privacy, and by integrating MPC techniques into the computations across multiple sources, I aim to achieve secure data and knowledge exchange and prevent leakage propagation. The efficiency and privacy guarantees of both systems should be theoretically constructed and empirically validated, through the collaboration with researchers from security and encryption.

REFERENCES

[1] C. Yang, J. Zhang, and J. Han, "Neural embedding propagation on heterogeneous networks," in *ICDM*, 2019.

[2] C. Yang, L. Gan, Z. Wang, J. Shen, J. Xiao, and J. Han, "Query-specific knowledge summarization with entity evolutionary networks," in *CIKM*, 2019.

[3] C. Yang, J. Zhang, H. Wang, B. Li, and J. Han, "Neural concept map generation for effective document classification with interpretable structured summarization," in *SIGIR*, 2020.

[4] C. Yang and K. Chang, "Relationship profiling over social networks: Reverse smoothness from similarity to closeness," in *SDM*, 2019.

[5] C. Yang, M. Liu, F. He, X. Zhang, J. Peng, and J. Han, "Similarity modeling on heterogeneous networks via automatic path discovery," in *ECML-PKDD*, 2018.

[6] C. Yang, M. Liu, V. W. Zheng, and J. Han, "Node, motif and subgraph: Leveraging network functional blocks through structural convolution," in *ASONAM*, 2018.

[7] C. Yang, Y. Feng, P. Li, Y. Shi, and J. Han, "Meta-graph based hin spectral embedding: Methods, analyses, and insights," in *ICDM*, 2018.

[8] C. Yang, J. Zhang, and J. Han, "Co-embedding network nodes and hierarchical labels with taxonomy based generative adversarial nets," in *ICDM*, 2020.

[9] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, "Conditional structure generation through graph variational generative adversarial nets," in *NIPS*, 2019.

[10] C. Yang, M. Liu, F. He, J. Peng, and J. Han, "cube2net: Efficient query-specific network construction with data cube organization," in *ICDM*, 2019.

[11] C. Yang, D. Teng, S. Liu, S. Basu, J. Zhang, J. Shen, C. Zhang, J. Shang, L. Kaplan, T. Harratty et al., "Cubenet: Multi-facet hierarchical heterogeneous network construction, analysis, and mining," in *KDD*, 2019.

[12] C. Yang, L. Liu, M. Liu, Z. Wang, C. Zhang, and J. Han, "Graph clustering with embedding propagation," in *BigData*, 2020.

[13] C. Yang, Y. Xiao, Y. Zhang, Y. Sun, and J. Han, "Heterogeneous network representation learning: A unified framework with survey and benchmark," in *TKDE*, 2020.

[14] C. Yang, A. Pal, A. Zhai, N. Pancha, , J. Han, C. Rosenberg, and J. Leskovec, "Multisage: Empowering graphsage with contextualized multi-embedding on web-scale multipartite networks," in *KDD*, 2020.

[15] C. Yang, X. Shi, L. Jie, and J. Han, "I know you'll be back: Interpretable new user clustering and churn prediction on a mobile social application," in *KDD*, 2018.

[16] C. Yang, J. Zhang, H. Wang, S. Li, M. Kim, M. Walker, Y. Xiao, and J. Han, "Relation learning on social networks with multi-modal graph edge variational autoencoders," in *WSDM*, 2020.

[17] C. Yang, D. H. Hoang, T. Mikolov, and J. Han, "Place deduplication with embeddings," in *WWW*, 2019.

[18] C. Yang, C. Zhang, X. Chen, J. Ye, and J. Han, "Did you enjoy the ride? understanding passenger experience via heterogeneous network embedding," in *ICDE*, 2018.

[19] C. Yang, L. Zhong, L.-J. Li, and L. Jie, "Bi-directional joint inference for user links and attributes on large social graphs," in *WWW*, 2017.

[20] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han, "Bridging collaborative filtering and semi-supervised learning: a neural approach for poi recommendation," in *KDD*, 2017.

[21] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, 2000.

[22] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, 2000.

[23] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *NIPS*, 2002.

[24] X. He and P. Niyogi, "Locality preserving projections," in *NIPS*, 2004.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.

[26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015.

[27] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*, 2015.

[28] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016.

[29] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014.

[30] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.

[31] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," in *AAAI*, 2018.

[32] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[33] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[34] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," in *ICLR*, 2018.

[35] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[36] Z. Yang, W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, 2016.

[37] H. Gao and H. Huang, "Self-paced network embedding," in *KDD*, 2018.

[38] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *KDD*, 2018.

[39] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *KDD*, 2017.

[40] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," in *AAAI*, 2018.

[41] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *KDD*, 2018.

[42] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information." in *IJCAI*, 2015.

[43] J. Tang, M. Qu, and Q. Mei, "Pte: Predictive text embedding through large-scale heterogeneous text networks," in *KDD*, 2015.

[44] S. Ganguly and V. Pudi, "Paper2vec: Combining graph and text information for scientific paper representation," in *ECIR*, 2017.

[45] J. Liu, Z. He, L. Wei, and Y. Huang, "Content to node: Self-translation network embedding," in *KDD*, 2018.

[46] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018.

[47] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017.

[48] Y. Shi, H. Gui, Q. Zhu, L. Kaplan, and J. Han, "Aspem: Embedding learning by aspects in heterogeneous information networks," in *SDM*, 2018.

[49] T. Chen and Y. Sun, "Task-guided and path-augmented heterogeneous network embedding for author identification," in *WSDM*, 2017.

[50] Y. Shi, Q. Zhu, F. Guo, C. Zhang, and J. Han, "Easing embedding learning by comprehensive transcription of heterogeneous information networks," in *KDD*, 2018.

[51] M. Qu, J. Tang, and J. Han, "Curriculum learning for heterogeneous star network embedding via deep reinforcement learning," in *WSDM*, 2018.

[52] H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, and J. Han, "Large-scale embedding learning in heterogeneous event data," in *ICDM*, 2016.

[53] H. Wang, F. Zhang, M. Hou, X. Xie, M. Guo, and Q. Liu, "Shine: Signed heterogeneous information network embedding for sentiment link prediction," in *WSDM*, 2018.

[54] M. Qu, J. Tang, J. Shang, X. Ren, M. Zhang, and J. Han, "An attention-based collaboration framework for multi-view network representation learning," in *CIKM*, 2017.

[55] T.-y. Fu, W.-C. Lee, and Z. Lei, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *CIKM*, 2017.

[56] C. Zhang, A. Swami, and N. V. Chawla, "Shne: Representation learning for semantic-associated heterogeneous networks," in *WSDM*, 2019.

[57] H. Zhang, L. Qiu, L. Yi, and Y. Song, "Scalable multiplex network embedding." in *IJCAI*, 2018.

[58] Y. Lu, C. Shi, L. Hu, and Z. Liu, "Relation structure-aware heterogeneous information network embedding," in *AAAI*, 2019.

[59] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *AAAI*, 2018.

[60] R. Hussein, D. Yang, and P. Cudré-Mauroux, "Are meta-paths necessary?: Revisiting heterogeneous graph embeddings," in *CIKM*, 2018.

[61] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *ESWC*, 2018.

[62] Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang, "Representation learning for attributed multiplex heterogeneous network," in *KDD*, 2019.

[63] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *WWW*, 2019.

[64] Y. Zhang, Y. Xiong, X. Kong, S. Li, J. Mi, and Y. Zhu, "Deep collective classification in heterogeneous information networks," in *WWW*, 2018.

[65] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *KDD*, 2019.

[66] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *TKDE*, 2017.

[67] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013.

[68] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014.

[69] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015.

[70] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *ACL*, 2015.

[71] G. Ji, K. Liu, S. He, and J. Zhao, "Knowledge graph completion with adaptive sparse transfer matrix," in *AAAI*, 2016.

[72] H. Xiao, M. Huang, and X. Zhu, "From one point to a manifold: Knowledge graph embedding for precise link prediction," *IJCAI*, 2015.

[73] J. Feng, M. Huang, M. Wang, M. Zhou, Y. Hao, and X. Zhu, "Knowledge graph embedding by flexible translation," in *ICKR*, 2016.

[74] H. Xiao, M. Huang, Y. Hao, and X. Zhu, "Transa: An adaptive approach for knowledge graph embedding," in *AAAI*, 2015.

[75] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data." in *ICML*, 2011.

[76] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *NIPS*, 2013.

[77] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *AAAI*, 2016.

[78] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," in *ICLR*, 2015.

[79] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *ICML*, 2016.

[80] H. Liu, Y. Wu, and Y. Yang, "Analogical inference for multi-relational embeddings," in *ICML*, 2017.

[81] X. Glorot, A. Bordes, J. Weston, and Y. Bengio, "A semantic matching energy function for learning with multi-relational data," *arXiv preprint arXiv:1301.3485*, 2013.

[82] F. Wu, J. Song, Y. Yang, X. Li, Z. Zhang, and Y. Zhuang, "Structured embedding via pairwise relations and long-range interactions in knowledge base," in *AAAI*, 2015.

[83] B. Shi and T. Weninger, "Proje: Embedding projection for knowledge graph completion," in *AAAI*, 2017.

[84] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *AAAI*, 2018.

[85] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, and B. Zhou, "End-to-end structure-aware convolutional networks for knowledge base completion," in *AAAI*, 2019.

[86] I. Balažević, C. Allen, and T. M. Hospedales, "Hypernetwork knowledge graph embeddings," in *IJCAI*, 2019.

[87] B. Oh, S. Seo, and K.-H. Lee, "Knowledge graph completion by context-aware convolutional learning with multi-hop neighborhoods," in *CIKM*, 2018.

[88] S. Guan, X. Jin, Y. Wang, and X. Cheng, "Shared embedding based neural networks for knowledge graph completion," in *CIKM*, 2018.

[89] K. Wang, Y. Liu, X. Xu, and D. Lin, "Knowledge graph embedding with entity neighbors and deep memory network," in *AAAI*, 2019.

[90] W. Qian, C. Fu, Y. Zhu, D. Cai, and X. He, "Translating embeddings for knowledge graph completion with relation attention mechanism." in *IJCAI*, 2018.

[91] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach," in *IJCAI*, 2017.

[92] P. Wang, J. Han, C. Li, and R. Pan, "Logic attention based neighborhood aggregation for inductive knowledge graph embedding," in *AAAI*, 2019.

[93] J. Feng, M. Huang, Y. Yang et al., "Gake: graph aware knowledge embedding," in *COLING*, 2016.

[94] Y. Cao, X. Wang, X. He, Z. Hu, and T.-S. Chua, "Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences," in *WWW*, 2019.

[95] T. Evans and R. Lambiotte, "Line graphs, link partitions, and overlapping communities," *Physical Review E*, 2009.

[96] D. Hallac, Y. Park, S. Boyd, and J. Leskovec, "Network inference via the time-varying graphical lasso," in *KDD*, 2017.

[97] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hungar. Acad. Sci*, 1960.

[98] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, 1998.

[99] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, 1999.

[100] M. E. Newman, "Clustering and preferential attachment in growing networks," *Physical review E*, 2001.

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

[102] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *ICML*, 2018.

[103] N. De Cao and T. Kipf, "Molgan: An implicit generative model for small molecular graphs," *arXiv preprint arXiv:1805.11973*, 2018.

[104] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *NIPS*, 2018.

[105] D. Zou and G. Lerman, "Encoding robust representation for graph generation," in *IJCNN*, 2019.

[106] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *ICML*, 2019.

[107] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018.

[108] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *ICANN*, 2018.

[109] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *ICML*, 2018.

[110] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," in *ICML*, 2018.

[111] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NIPS*, 2018.

[112] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec, "Pixie: A system for recommending 3+ billion items to 200+ million users in real-time," in *WWW*, 2018.

[113] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *KDD*, 2008.

[114] D. Chakrabarti, S. Funiak, J. Chang, and S. A. Macskassy, "Joint inference of multiple label types in large networks," in *ICML*, 2014.

[115] R. Li, C. Wang, and K. C.-C. Chang, "User profiling in an ego network: co-profiling attributes and relationships," in *WWW*, 2014.

[116] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[117] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*, 2010.

[118] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *KDD*, 2015.

[119] X. Ren, A. El-Kishky, C. Wang, F. Tao, C. R. Voss, and J. Han, "Clustype: Effective entity recognition and typing by relation phrase-based clustering," in *KDD*, 2015.

[120] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," in *NAACL*, 2016.

[121] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," in *ACL*, 2016.

[122] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," in *ACL*, 2016.

[123] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin, "Classifying relations via long short term memory networks along shortest dependency paths," in *EMNLP*, 2015.

[124] L. Liu, X. Ren, Q. Zhu, S. Zhi, H. Gui, H. Ji, and J. Han, "Heterogeneous supervision for relation extraction: A representation learning approach," in *EMNLP*, 2017.

[125] E. Agichtein and L. Gravano, "Snowball: Extracting relations from large plain-text collections," in *CDL*, 2000.

[126] M. Qu, X. Ren, and J. Han, "Automatic synonym discovery with knowledge bases," in *KDD*, 2017.

[127] M. Jiang, J. Shang, T. Cassidy, X. Ren, L. M. Kaplan, T. P. Hanratty, and J. Han, "Metapad: Meta pattern discovery from massive text corpora," in *KDD*, 2017.

[128] R. Leaman and G. Gonzalez, "Banner: an executable survey of advances in biomedical named entity recognition," in *Biocomputing*, 2008.

[129] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *ACL*, 2005.

[130] J. Ebrahimi and D. Dou, "Chain based rnn for relation classification," in *ACL*, 2015.

[131] Q. Li and H. Ji, "Incremental joint extraction of entity mentions and relations," in *ACL*, 2014.

[132] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *ACL*, 2016.

[133] X. Ren, Z. Wu, W. He, M. Qu, C. R. Voss, H. Ji, T. F. Abdelzaher, and J. Han, "Cotype: Joint extraction of typed entities and relations with knowledge bases," in *WWW*, 2017.

[134] Z. Wu, X. Ren, F. F. Xu, J. Li, and J. Han, "Indirect supervision for relation extraction using question-answer pairs," in *WSDM*, 2018.

[135] Y. Xu, R. Jia, L. Mou, G. Li, Y. Chen, Y. Lu, and Z. Jin, "Improved relation classification by deep recurrent neural networks with data augmentation," in *COLING*, 2016.

[136] J. Shang, L. Liu, X. Ren, X. Gu, and J. Han, "Learning named entity tagger using domain-specific dictionary," in *EMNLP*, 2018.

[137] L. Liu, J. Shang, X. Ren, F. F. Xu, H. Gui, J. Peng, and J. Han, "Empower sequence labeling with task-aware neural language model," in *AAAI*, 2018.

[138] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *COLING*, 1992.

[139] S. P. Ponzetto and M. Strube, "Deriving a large scale taxonomy from wikipedia," in *AAAI*, 2007.

[140] J. Seitner, C. Bizer, K. Eckert, S. Faralli, R. Meusel, H. Paulheim, and S. P. Ponzetto, "A large database of hypernymy relations extracted from the web." in *LREC*, 2016.

[141] N. Nakashole, G. Weikum, and F. Suchanek, "Patty: a taxonomy of relational patterns with semantic types," in *EMNLP*, 2012.

[142] M. Bansal, D. Burkett, G. De Melo, and D. Klein, "Structured learning for taxonomy induction with belief propagation," in *ACL*, 2014.

[143] T. L. Anh, Y. Tay, S. C. Hui, and S. K. Ng, "Learning term embeddings for taxonomic relation identification using dynamic weighting neural network," in *EMNLP*, 2016.

[144] R. L. de Mantaras and L. Saitia, "Comparing conceptual, divisive and agglomerative clustering for learning taxonomies from text," in *ECAI*, 2004.

[145] Z. Kozareva and E. Hovy, "A semi-supervised method to learn and construct taxonomies using the web," in *EMNLP*, 2010.

[146] X. Liu, Y. Song, S. Liu, and H. Wang, "Automatic taxonomy construction from keywords," in *KDD*, 2012.

[147] H. Yang and J. Callan, "A metric-based framework for automatic taxonomy induction," in *ACL*, 2009.

[148] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," in *NIPS*, 2002.

[149] X. Zhu, Z. Ghahramani, J. Lafferty et al., "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, 2003.

[150] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*, 2012.

[151] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," in *ICML*, 2006.

[152] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang, "Manifold-ranking based image retrieval," in *Multimedia*, 2004.

[153] D. F. Gleich and M. W. Mahoney, "Using local spectral methods to robustify graph-based learning algorithms," in *KDD*, 2015.

[154] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NIPS*, 2018.

[155] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," in *VLDB*, 2011.

[156] X. Yu, Y. Sun, B. Norick, T. Mao, and J. Han, "User guided entity similarity search using meta-path selection in heterogeneous information networks," in *CIKM*, 2012.

[157] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in *Recommender systems handbook*, 2011.

[158] Y. Liu, T.-A. N. Pham, G. Cong, and Q. Yuan, "An experimental evaluation of point-of-interest recommendation in location-based social networks," in *VLDB*, 2017.

[159] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *RecSys*, 2010.

[160] X. Geng, H. Zhang, J. Bian, and T.-S. Chua, "Learning image and user features for recommendation in social networks," in *ICCV*, 2015.

[161] B. Liu, Y. Fu, Z. Yao, and H. Xiong, "Learning geographical preferences for point-of-interest recommendation," in *KDD*, 2013.

[162] J.-D. Zhang and C.-Y. Chow, "igslr: personalized geo-social location recommendation: a kernel density estimation approach," in *SIGSPATIAL*, 2013.

[163] C. Shi, B. Hu, W. X. Zhao, and S. Y. Philip, "Heterogeneous information network embedding for recommendation," *TKDE*, 2018.

[164] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003.

[165] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *KDD*, 2002.

[166] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *NIPS*, 2014.

[167] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *WSDM*, 2018.

[168] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[169] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with gnns," in *NIPS*, 2019.

[170] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," in *NIPS*, 2019.

[171] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *ICDM*, 2005.

[172] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning theory and kernel machines*, 2003.

[173] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," in *ICML*, 2011.

[174] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *AIStats*, 2009.

[175] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *KDD*, 2015.

[176] J. Shang, J. Liu, M. Jiang, X. Ren, C. R. Voss, and J. Han, "Automated phrase mining from massive text corpora," *TKDE*, 2018.

[177] C. Zhang, F. Tao, X. Chen, J. Shen, M. Jiang, B. Sadler, M. Vanni, and J. Han, "Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering," in *KDD*, 2018.

[178] F. Tao, H. Zhuang, C. W. Yu, Q. Wang, T. Cassidy, L. M. Kaplan, C. R. Voss, and J. Han, "Multi-dimensional, phrase-based summarization in text cubes," *IEEE Data Eng. Bull.*, 2016.

[179] J. Shen, J. Xiao, X. He, J. Shang, S. Sinha, and J. Han, "Entity set search of scientific literature: An unsupervised ranking approach," in *SIGIR*, 2018.

[180] B. Ding, B. Zhao, C. X. Lin, J. Han, C. Zhai, A. Srivastava, and N. C. Oza, "Efficient keyword-based search for top-k cells in text cube," *TKDE*, 2011.

[181] P. Zhao, X. Li, D. Xin, and J. Han, "Graph cube: on warehousing and olap multidimensional networks," in *SIGMOD*, 2011.

[182] X. Yan and J. Han, "Closegraph: mining closed frequent graph patterns," in *KDD*, 2003.

[183] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *TPAMI*, 2013.

[184] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Label-embedding for image classification," *TPAMI*, 2016.

[185] Y. Ma, E. Cambria, and S. Gao, "Label embedding for zero-shot fine-grained named entity typing," in *COLING*, 2016.

[186] G. Wang, C. Li, W. Wang, Y. Zhang, D. Shen, X. Zhang, R. Henao, and L. Carin, "Joint embedding of words and labels for text classification," in *ACL*, 2018.

[187] M. Alsuhaibani, T. Maehara, and D. Bollegala, "Joint learning of hierarchical word embeddings from a corpus and a taxonomy," in *AKBC*, 2018.

[188] K. A. Nguyen, M. Köper, S. S. im Walde, and N. T. Vu, "Hierarchical embeddings for hypernymy detection and directionality," in *EMNLP*, 2017.

[189] I. Vulić and N. Mrkšić, "Specialising word vectors for lexical entailment," in *ACL*, 2018.

[190] Z. Yu, H. Wang, X. Lin, and M. Wang, "Learning term embeddings for hypernymy identification," in *IJCAI*, 2015.

[191] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-cnn," in *WWW*, 2018.

[192] J. Wehrmann, R. Cerri, and R. Barros, "Hierarchical multi-label classification networks," in *ICML*, 2018.

[193] Q. Dai, X. Shen, L. Zhang, Q. Li, and D. Wang, "Adversarial training methods for network embedding," in *WWW*, 2019.

[194] H. Gao, J. Pei, and H. Huang, "Progan: Network embedding via proximity generative adversarial network," in *KDD*, 2019.

[195] Z. Meng, S. Liang, H. Bao, and X. Zhang, "Co-embedding attributed networks," in *WSDM*, 2019.

[196] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *SDM*, 2017.

[197] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang, "Anrl: Attributed network representation learning via deep neural networks." in *IJCAI*, 2018.

[198] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *WSDM*, 2017.

[199] M. Nickel and D. Kiela, "Poincaré embeddings for learning hierarchical representations," in *NIPS*, 2017.

[200] J. Ma, P. Cui, X. Wang, and W. Zhu, "Hierarchical taxonomy aware network embedding," in *KDD*, 2018.

[201] N. Liu, X. Huang, J. Li, and X. Hu, "On interpretation of network embedding via taxonomy induction," in *KDD*, 2018.

[202] J. Park, B. J. Hescott, and D. K. Slonim, "Towards a more molecular taxonomy of disease," *JBS*, 2017.

[203] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie, "Stacked generative adversarial networks," in *CVPR*, 2017.

[204] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *AAAI*, 2017.

[205] H. Maron, H. Ben-Hamu, N. Sharmir, and L. Yaron, "Invariant and equivariant graph networks," in *ICLR*, 2019.

[206] J. Hartford, D. R. Graham, K. Leyton-Brown, and S. Ravanbakhsh, "Deep models of interactions across sets," in *ICML*, 2018.

[207] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018.

[208] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," in *ICLR*, 2019.

[209] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *CVPR*, 2014.

[210] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018.

[211] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," in *ICML*, 2010.

[212] G. Robins, P. Pattison, Y. Kalish, and D. Lusher, "An introduction to exponential random graph (p*) models for social networks," *Social networks*, 2007.

[213] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *NIPS*, 2015.

[214] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[215] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *ICML*, 2016.

[216] X. Gu, K. Cho, J. Ha, and S. Kim, "Dialogwae: Multimodal response generation with conditional wasserstein auto-encoder," in *ICLR*, 2019.

[217] M. Rosca, B. Lakshminarayanan, D. Warde-Farley, and S. Mohamed, "Variational approaches for auto-encoding generative adversarial networks," *arXiv preprint arXiv:1706.04987*, 2017.

[218] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[219] Y. Bai, T. Ma, and A. Risteski, "Approximability of discriminators implies diversity in gans," in *ICLR*, 2019.

[220] J. Du, P. Jia, Y. Dai, C. Tao, Z. Zhao, and D. Zhi, "Gene2vec: distributed representation of genes based on co-expression," *BMC Genomics*, 2019.

[221] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *TKDE*, 2018.

[222] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *NIPS*, 2018.

[223] S. Verma and Z.-L. Zhang, "Stability and generalization of graph convolutional neural networks," in *KDD*, 2019.

[224] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *KDD*, 2019.

[225] J. Klicpera, A. Bojchevski, and S. Günnemann, "Combining neural networks with personalized pagerank for classification on graphs," in *ICLR*, 2019.

[226] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016.

[227] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *ACHA*, 2011.

[228] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, 2018.

[229] J. Zhao, Z. Zhou, Z. Guan, W. Zhao, W. Ning, G. Qiu, and X. He, "Intentgc: a scalable graph convolution framework fusing heterogeneous information for recommendation," in *KDD*, 2019.

[230] S. Fan, J. Zhu, X. Han, C. Shi, L. Hu, B. Ma, and Y. Li, "Metapath-guided heterogeneous graph neural network for intent recommendation," in *KDD*, 2019.

[231] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. S. Dickstein, "On the expressive power of deep neural networks," in *ICML*, 2017.

[232] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[233] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, 2013.

[234] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, "Sampling matters in deep embedding learning," in *ICCV*, 2017.

[235] C. C. Aggarwal, *Data streams: models and algorithms*. Springer, 2007.

[236] N. Harris, *Method for customizing multi-media advertisement for targeting specific demographics*. US Patent App., 2006.

[237] S. Moriarty, N. D. Mitchell, W. D. Wells, R. Crawford, L. Brennan, and R. Spence-Stone, *Advertising: Principles and practice*. Pearson Australia, 2014.

[238] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *JCAM*, 1987.

[239] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[240] M. Denil, L. Bazzani, H. Larochelle, and N. de Freitas, "Learning where to attend with deep architectures for image tracking," *Neural computation*, 2012.

[241] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[242] S. Ceyhan, X. Shi, and J. Leskovec, "Dynamics of bidding in a p2p lending service: effects of herding and predicting loan success," in *WWW*, 2011.

[243] X. Shi, M. Bonner, L. A. Adamic, and A. C. Gilbert, "The very small world of the well-connected," in *HT*, 2008.

[244] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[245] Y. Hu, L. Manikonda, S. Kambhampati et al., "What we instagram: A first analysis of instagram photo content and user types." in *ICWSM*, 2014.

[246] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *WWW*, 2010.

[247] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *IJDWM*, 2006.

[248] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *JMLR*, 2008.