ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

ISSN:2229-6107

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY

E-mail :
editor.ijpast@gmail.com
editor@ijpast.in

www.ijpast.in

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY
IJPAST

ISSN 2229-6107 www.ijpast.in
Vol 9,Issue 1.Jan 2019

A Model and Extensive Taxonomy for Machine Learning on Graphs

Y. Prasada Reddy

Abstract

Interest in graph representation learning (GRL) has recently skyrocketed. In general, there are three broad types of GRL approaches that have developed in response to the availability of labeled data. The first one is network embedding, which is all about learning relational structure representations without supervision. The second one is called graph regularized neural networks, and it uses graphs to teach semi-supervised learning by adding a regularization goal to neural network losses. Finally, graph neural networks are designed to learn differentiable functions across arbitrary-structured discrete topologies. Interestingly, however, there has been relatively no effort to integrate the three paradigms, even though these fields are somewhat popular. Here, we strive to connect graph neural networks, graph regularization, and network embedding. In an effort to bring together several separate areas of study, we provide a thorough taxonomy of GRL approaches. In particular, we suggest the GRAPHEDM framework, which unifies well- known methods for learning graph representations using semi-supervised (e.g. GraphSage, GCN, GAT) and unsupervised (e.g., DeepWalk, node2vec) means. We fitted more than thirty existing techniques into this framework to demonstrate GRAPHEDM's generalizability. We think this unified perspective does double duty: it lays the groundwork for future study in the field and helps us comprehend the thinking underlying these techniques.

Keywords: Learning on Manifolds, Relational Learning, Geometric Deep Learning, and Network Embedding

Introduction

Developing representations for intricate structured data sets is no easy feat. Data defined on a discretized Euclidean domain is one kind of structured data that has seen a plethora of effective models produced in the last ten years.

One example is the use of recurrent neural networks for modeling sequential data, like text or movies. These networks are able to collect sequential

Assistant Professor
Department of Computer Science and Engineering
KSRM College of Engineering (A) Kadapa

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

information and provide efficient representations, as shown by their performance on machine translation and voice recognition tasks. Convolutional neural networks (CNNs) are another example; they have achieved remarkable performance in pattern recognition tasks like image classification and voice recognition by parameterizing neural networks according to structural priors like shift- invariance. These remarkable achievements have only been applicable to certain kinds of data with a straightforward relational structure, such as sequential data or data that follows regular patterns. Data is not always so regular; complex relationship structures often emerge, and comprehending the interplay between objects requires data extraction from such systems. Social networks, computational chemistry, biology, recommendation systems, semi-supervised learning, and other domains make use of graphs, which are universal data structures that can represent complex relational data (made up of nodes and edges) (Gilmer et al., 2017; Stark et al., 2006; Konstas et al., 2009; Garcia and Bruna, 2018). Since graph topologies are not always consistent and may change greatly across graphs and even between nodes in the same graph, it is difficult to construct networks with strong structural priors for graph-structured data. Irregular graph domains are particularly incompatible with operations like convolutions. For example, since all of the pixels in an image have the same neighborhood structure, it is possible to use the same filter weights everywhere in the picture. Nevertheless, given that every node in a network may have a unique neighborhood structure, it is impossible to provide an ordering of nodes (Fig. 1). On top of that, non- Euclidean domains are not applicable to geometric priors (such as shift invariance) used in Euclidean convolutions (for instance, translations may not even be specified on such domains).

Research into Geometric Deep Learning (GDL) emerged in response to these difficulties; GDL seeks to apply deep learning methods to data that is not geometrically normal. A lot of people are very interested in using machine learning techniques on graph-structured data because of how common graphs are in real-world applications.Learned embeddings are low-dimensional continuous vector representations of graph- structured data; GRL techniques are one such approach. Unsupervised GRL and supervised (or semi-supervised) GRL are the two main categories of GRL learning tasks. The first set of rules is based on the notion of learning low-dimensional Euclidean representations that retain the original graph structure. For a particular downstream prediction job, such node or graph categorization, the second family likewise learns low-dimensional Euclidean representations. In contrast to the unsupervised environment, whereby inputs are often graph structures, the supervised setting typically uses a variety of signals specified on graphs, or node attributes, as inputs. Whereas in the inductive learning scenario, the underlying discrete graph domain may change (for example, when predicting molecular attributes where each molecule is a graph), in the transductive learning context, it can remain stable (for example, when predicting user qualities in a huge social network). Lastly, it should be mentioned that the majority of supervised and unsupervised approaches learn representations in vector spaces that are based on geometry, but there has been a recent uptick in interest in non-Euclidean representation learning. This kind of learning attempts to acquire knowledge about embedding spaces that are not based on geometry, such as spherical or hyperbolic spaces. The primary goal of this research is to use an embedding space that is continuous and similar to the input data's underlying discrete structure (for instance, hyperbolic space is a continuous form of trees; Sarkar, 2011).

We think it is critical to synthesize and explain these techniques in one cohesive and understandable framework since the GRL field is expanding at a remarkable rate. This review aims to provide a comprehensive overview of representation learning techniques for graph- structured data so that readers may have a better understanding of the many ways in which deep learning models use graph structure.

There are an assortment of graph representation learning questionnaires available. For a full review of shallow network embedding and auto-encoding approaches, there are various surveys that address the topic. We recommend (Cai et al., 2018;Chen et al., 2018a; Goyal and Ferrara, 2018b; Hamilton et al., 2017b; Zhang et al., 2018a) for this. Second, for data that is not Euclidean, such manifolds or graphs, Bronstein et al. (2017) provides a comprehensive review of deep learning methods. Thirdly, approaches applying deep learning to graphs, particularly graph neural networks, have been covered in many recent surveys (Battaglia et al., 2018; Wu et al., 2019; Zhang et al., 2018c; Zhou et al., 2018). Rather than establishing links across several areas of graph representation learning, most of these studies focus down on only one.

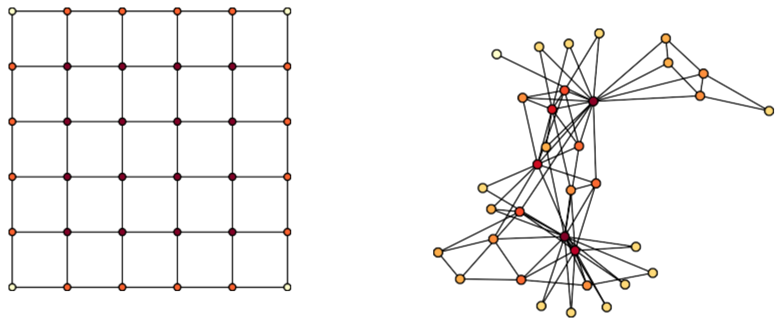
We develop a general framework called the Graph

Encoder Decoder Model (GRAPHEDM) to classify previous work into four main areas: (i) methods for shallow embedding, (ii) methods for auto-encoding, (iii) methods for graph regularization, and (iv) methods for graph neural networks (GNNs). This framework expands upon the encoder-decoder model proposed by Hamilton et al. (2017b).We also provide a Graph Convolution Framework (GCF) for describing convolution-based GNNs, which have shown to be very effective in many different domains. According to Veliˇckovi'c et al. (2018), we are able to examine and contrast several GNNs, which differ in their design. These GNNs range from those that operate in the Graph Fourier1 domain to those that use self-attention as a neighborhood aggregation function. The goal of this comprehensive synthesis of current research is to provide readers with a better understanding of the many graph-based learning approaches so that they may identify their -

similarities and differences, as well as their possible expansions and limits. However, there are three ways in which our survey differs from earlier ones:

We introduce a general framework, GRAPHEDM, to describe a broad range of super- vised and unsupervised methods that operate on graph-structured data, namely shal- low embedding methods, graph regularization methods, graph auto- encoding methods and graph neural networks.

Our survey is the first attempt to unify and view these different lines of work from the same perspective, and we provide a general taxonomy (Fig. 3) to understand differences and similarities between these methods. In particular, this taxonomy en



(a) Grid (Euclidean). (b) Arbitrary graph (Non-Euclidean).

Figure 1: An illustration of Euclidean vs. non-Euclidean graphs.

represents more than 30 different GRL algorithms. To better understand the differences between various strategies, it is helpful to describe them within a thorough taxonomy.

• We provide an open-source GRL library that contains cutting-edge GRL methods and crucial graph applications including link prediction and node categorization. You may find our implementation is open to the public.

Organization of the survey Section 2 provides a clear statement of the issue setting for GRL and a review of fundamental graph concepts. Section 2.2.1 explains the function of node features in GRL and their relationship to supervised GRL; Section 2.2.2 differentiates between inductive and transductive learning; Section 2.2.3.1 distinguishes between positional and structural embeddings; and Section 2.2.4 distinguishes between supervised and unsupervised embeddings. We also define and discuss the differences between these important concepts in GRL. Section 3 then presents GRAPHEDM, a generic framework that may be used in inductive and transductive learning contexts to define supervised and unsupervised GRL techniques, with or without node characteristics. We provide a comprehensive taxonomy of GRL approaches (Fig. 3) based on GRAPHEDM, which incorporates more than thirty contemporary GRL models. We use this taxonomy to characterize both supervised (Section 5) and unsupervised (Section 4) methods. Section 6 concludes with an overview of graph applications.

1. Preliminaries

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY IJPAST

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

2. Graph representation learning approaches attempt to address the generalized network embedding issue; for an overview, see Table 1. Here, we offer the notation used throughout the article.

	Notation	Meaning
	GRL	Graph Representation
Abbreviations	GRAPHED	Learning Graph Encoder Decoder
	MGNN	Model Graph Neural Network
	GCF	Graph Convolution Framework
	$G = (V, E)$	Graph with vertices (nodes) V and edges E
	$v_i \in V$	Graph vertex
	$d_G(\cdot, \cdot)$ $\deg(\cdot)$	Graph distance (length of shortest path) Node
		degree
	$D \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	Diagonal degree matrix
Graph notation	$W \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$ $\widetilde{W} \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	Graph weighted adjacency matrix
		Symmetric normalized adjacency matrix ($\widetilde{W} = D^{-1/2}WD^{-1/2}$) Graph
	$A \in \{0, 1\}^{\lvert V\rvert \times \lvert V\rvert}$	unweighted weighted adjacency matrix
	$L \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$ $\tilde{L} \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	Graph unnormalized Laplacian matrix ($L = D - W$) Graph
	$L^{\mathrm{rw}} \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	normalized Laplacian matrix ($\tilde{L} = I - D^{-1/2}WD^{-1/2}$) Random walk
		normalized Laplacian ($L^{\mathrm{rw}} = I - D^{-1}W$)
	d_0	Input feature dimension
	$X \in \mathbb{R}^{\lvert V\rvert \times d_0}$	Node feature matrix
	d	Final embedding dimension
	$Z \in \mathbb{R}^{\lvert V\rvert \times d}$	Node embedding matrix
	d_l	Intermediate hidden embedding dimension at layer l
	$H^l \in \mathbb{R}^{\lvert V\rvert \times d_{Æ}}$	Hidden representation at layer l
	$\mathcal{Y}$	Label space
	$y^S \in \mathbb{R}^{\lvert V\rvert \times \lvert\mathcal{Y}\rvert}$ $\hat{y}^S \in \mathbb{R}^{\lvert V\rvert \times \lvert\mathcal{Y}\rvert}$	Graph ($S = G$) or node ($S = N$) ground truth labels Predicted labels
GRAPHEDM notation	$s(W) \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$ $\widehat{W} \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	Target similarity or dissimilarity matrix in graph regularization Predicted similarity or dissimilarity matrix
	$\mathrm{ENC}(\cdot; \Theta^E)$	Encoder network with parameters Θ^E Graph
	$\mathrm{DEC}(\cdot; \Theta^D)$	decoder network with parameters Θ^D Label
	$\mathrm{DEC}(\cdot; \Theta^S)$	decoder network with parameters Θ^S Supervised
	$\mathcal{L}^S_{\mathrm{SUP}}(y^S, \hat{y}^S; \Theta)$	loss
	$\mathcal{L}_{G,\mathrm{REG}}(W, \widehat{W}; \Theta)$	Graph regularization loss
	$\mathcal{L}_{\mathrm{REG}}(\Theta)$	Parameters' regularization loss
	$d_1(\cdot, \cdot)$	Matrix distance used for to compute the graph regularization loss
	$d_2(\cdot, \cdot)$	Embedding distance for distance-based decoders
	$\lVert \cdot \rVert_p$	p–norm Frobenuis
	$\lVert \cdot \rVert_F$	norm

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

2.1 Definitions

Table 1: Summary of the notation used in the paper.

Definition 1 *(Graph). A graph G given as a pair: G = (V, E), comprises a set of vertices (or nodes) V = $\{v_1, \dots, v_{|V|}\}$ connected by edges E = $\{e_1, \dots, e_{|E|}\}$, where each edge e_k is a pair (v_i, v_j) with $v_i, v_j \in V$. A graph is weighted if there exist a weight function: $w : (v_i, v_j) \to w_{ij}$ that assigns weight w_{ij} to edge connecting nodes $v_i, v_j \in V$. Otherwise, we say that the graph is unweighted. A graph is undirected if $(v_i, v_j) \in E$ implies $(v_j, v_i) \in E$, i.e. the relationships are symmetric, and directed if the existence of edge $(v_i, v_j) \in E$ does not necessarily imply $(v_j, v_i) \in E$. Finally, a graph can be homogeneous if nodes refer to one type of entity and edges to one relationship. It can be heterogeneous if it contains different types of nodes and edges.*

For instance, social networks are homogeneous graphs that can be undirected (e.g. to encode symmetric relations like friendship) or directed (e.g. to encode the relation following); weighted (e.g. co-activities) or unweighted.

Definition 2 *(Path). A path P is a sequence of edges $(u_{i1}, u_{i2}), (u_{i2}, u_{i3}), \dots, (u_{ik}, u_{ik+1})$ of length k. A path is called simple if all u_{ij} are distinct from each other. Otherwise, if a path visits a node more than once, it is said to contain a cycle.*

Definition 3 *(Distance). Given two nodes (u, v) in a graph G, we define the distance from u to v, denoted $d_G(u, v)$, to be the length of the shortest path from u to v, or ∞ if there exist no path from u to v.*

The graph distance between two nodes is the analog of geodesic lengths on manifolds.

Definition 4 *(Vertex degree). The degree, $\deg(v_i)$, of a vertex v_i in an unweighted graph is the number of edges incident to it. Similarly, the degree of a vertex v_i in a weighted graph is the sum of incident edges weights. The degree matrix D of a graph with vertex set V is the $|V| \times |V|$ diagonal matrix such that $D_{ii} = \deg(v_i)$.*

Definition 5 *(Adjacency matrix). A finite graph G = (V, E) can be represented as a square $|V| \times |V|$ adjacency matrix, where the elements of the matrix indicate whether pairs of nodes are adjacent or not. The adjacency matrix is binary for unweighted graph, $A \in \{0, 1\}^{|V| \times |V|}$, and non-binary for weighted graphs $W \in \mathbb{R}^{|V| \times |V|}$. Undirected graphs have symmetric adjacency matrices, in which case, $\widetilde{\ }$ denotes ymmetrically-normalized adjacency matrix: $W = D^{-1/2}WD^{-1/2}$, where D is the degree matrix.*

Definition 6 *(Laplacian). The unnormalized Laplacian of an u~ndirected graph is the $|V| \times |V|$ matrix $L = D - W$. The symmetric normalized Laplacian is $L = I - D^{-1/2}WD^{-1/2}$. The random walk normalized Laplacian is the matrix $L^{rw} = I - D^{-1}W$.*

The name random walk comes from the fact that $D^{-1}W$ is a stochastic transition matrix that can be interpreted as the transition probability matrix of a random walk on the graph. The graph Laplacian is a key operator on graphs and can be interpreted as the analogue of the continuous Laplace-Beltrami operator on manifolds. Its eigenspace capture important properties about a graph (e.g. cut information often used for spectral graph clustering) but can also serve as a basis for smooth functions defined on the graph for semi-supervised learning (Belkin and Niyogi, 2004). The graph Laplacian is also closely related to the heat equation on graphs as it is the generator of diffusion processes on graphs and can be used to derive algorithms for semi- supervised learning on graphs (Zhou et al., 2004).

Definition 7 *(First order proximity). The first order proximity between two nodes v_i and v_j is a local similarity measure indicated by the edge weight w_{ij}. In other words, the first- order proximity captures the strength of an edge between node v_i and node v_j (should it exist).*

Definition 8 *(Second-order proximity). The second order proximity between two nodes v_i and v_j is measures the similarity of their neighborhood structures. Two nodes in a network will have a high second-order proximity if they tend to share many neighbors.*

Note that there exist higher-order measures of proximity between nodes such as Katz Index, Adamic Adar or Rooted PageRank (Liben-Nowell and Kleinberg, 2007). These notions of

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

node proximity are particularly important in network embedding as many algorithms are optimized to preserve some order of node proximity in the graph.

The generalized network embedding problem *Network embedding* is the task that aims at learning a mapping function from a discrete graph to a continuous domain. Formally, given a graph $G = (V, E)$ with weighted adjacency matrix $W \in \mathbb{R}^{|V|\times|V|}$, the goal is to learn low-dimensional vector representations $\{Z_i\}_{i\in V}$ (embeddings) for nodes in the graph $\{v_i\}_{i\in V}$, such that important graph properties (e.g. local or global structure) are preserved in the embedding space. For instance, if two nodes have similar connections in the original graph, their learned vector representations should be close. Let $Z \in \mathbb{R}^{|V|\times d}$ denote the node[2] embedding matrix. In practice, we often want low-dimensional embeddings ($d \quad |V|$) for scalability purposes. That is, network embedding can be viewed as a dimensionality reduction technique for graph structured data, where the input data is defined on a non- Euclidean, high-dimensional, discrete domain.

NODE FEATURES IN NETWORK EMBEDDING

Definition 9 *(Vertex and edge fields). A vertex field is a function defined on vertices $f : V \to \mathbb{R}$ and similarly an edge field is a function defined on edges: $F : E \to \mathbb{R}$. Vertex fields and edge fields can be viewed as analogs of scalar fields and tensor fields on manifolds.* Graphs may have node attributes (e.g. gender or age in social networks; article contents for citation networks) which can be represented as multiple vertex fields, commonly referred to as *node features*. In this survey, we denote node features with $X \in \mathbb{R}^{|V|\times d0}$, where d_0 is the input feature dimension. Node features might provide useful information about a graph. Some network embedding algorithms leverage this information by learning mappings:
$W, X \to Z.$
In other scenarios, node features might be unavailable or not useful for a given task: net- work embedding can be *featureless*. That is, the goal is to learn graph representations via mappings:
$W \to Z.$
Although we present the model taxonomy via embedding nodes yielding $Z \in \mathbb{R}^{|V|\times d}$, it can also be extended for models that embed an entire graph i.e. with $Z \in \mathbb{R}^d$ as a d-dimensional vector for the whole graph (e.g. (Duvenaud et al., 2015; Al-Rfou et al., 2019)), or embed graph edges $Z \in \mathbb{R}^{|V|\times|V|\times d}$ as a (potentially sparse) 3D matrix with $Z_{u,v} \in \mathbb{R}^d$ representing the embedding of edge (u, v). Note that depending on whether node features are used or not in the embedding algorithm, the learned representation could capture different aspects about the graph. If nodes features are being used, embeddings could capture both *structural* and *semantic* graph information. On the other hand, if node features are not being used, embeddings will only preserve structural information of the graph.
Finally, note that edge features are less common than node features in practice, but can also be used by embedding algorithms. For instance, edge features can be used as regularization for node embeddings (Chen et al., 2018c), or to compute messages from neighbors as in message passing networks (Gilmer et al., 2017).

TRANSDUCTIVE AND INDUCTIVE NETWORK EMBEDDING

Historically, a popular way of categorizing a network embedding method has been by whether the model can generalize to unseen data instances – methods are referred to as operating in either a *transductive* or *inductive* setting (Yang et al., 2016). While we do not use this concept for constructing our taxonomy, we include a brief discussion here for completeness.
In transductive settings, it assumed that all nodes in the graph are observed in training (typically the nodes all come from one fixed graph). These methods are used to infer information about or between observed nodes in the graph (e.g. predicting labels for all nodes, given a partial labeling). For instance, if a transductive method is used to embed the nodes of a social network, it can be used to suggest new edges (e.g. friendships) between the nodes of the graph. One major limitation of models learned in transductive settings is that they fail to generalize to new nodes (e.g. evolving graphs) or new graph instances.
On the other hand, in inductive settings, models are expected to generalize to new nodes, edges, or graphs that were not observed during training. Formally, given training graphs $(G_1, \ldots, G_k)$, the goal is to learn a mapping to continuous representations that can generalize to unseen test graphs $(G_{k+1}, \ldots, G_{k+l})$. For instance, inductive learning can be used to embed molecular graphs, each representing a molecule structure (Gilmer et al., 2017), generalizing to new graphs and showing error margins within chemical accuracy on many quantum properties. Embedding dynamic or temporally evolving graphs is also another inductive graph embedding problem.

There is a strong connection between inductive graph embedding and *node features* (Sec- tion 2.2.1) as the latter are usually necessary for most inductive graph representation learn- ing algorithms. More concretely, node features can be leveraged to learn embeddings with parametric mappings and instead of directly optimizing the embeddings, one can optimize the mapping's parameters. The learned mapping can then be applied to any node (even those that were not present a training time). On the other hand, when node features are not available, the first mapping from nodes to embeddings is usually a one-hot encoding which fails to generalize to new graphs where the canonical node ordering is not available.

Finally, we note that this categorization of graph embedding methods is at best an incomplete lens for viewing the landscape. While some models are inherently better suited to different tasks in practice, recent theoretical results (Srinivasan and Ribeiro, 2020) show that models previously assumed to be capable of only one setting (e.g. only transductive) can be used in both.

Positional vs structural network embedding

An emerging categorization of graph embedding algorithms is about whether the learned embeddings are positional or structural. Position-aware embeddings capture global relative positions of nodes in a graph and it is common to refer to embeddings as positional if they can be used to approximately reconstruct the edges in the graph, preserving distances such as shortest paths in the original graph (You et al., 2019). Examples of positional embedding algorithms include random walk or matrix factorization methods. On the other hand, structure-aware embeddings capture local structural information about nodes in a graph, i.e. nodes with similar node features or similar structural roles in a network should have similar embeddings, regardless of how far they are in the original graph. For instance, GNNs usually learn embeddings by incorporating information for each node's neighborhood, and the learned representations are thus structure-aware.

In the past, positional embeddings have commonly been used for unsupervised tasks where positional information is valuable (e.g. link prediction or clustering) while structural embeddings have been used for supervised tasks (e.g. node classification or whole graph classification). More recently, there has been attempts to bridge the gap between positional and structural representations, with positional GNNs (You et al., 2019) and theoretical frameworks showing the equivalence between the two classes of embeddings (Srinivasan and Ribeiro, 2020).

Unsupervised and supervised network embedding

Depending on whether extra information like node or graph labels is supplied, network embedding may be either supervised or unsupervised. The former case involves using simply the graph structure and, in certain cases, node attributes. Optimization of a reconstruction loss—a measure of the learnt embeddings' ability to mimic the original graph—is often used in unsupervised network embedding with the objective of learning embeddings that retain the graph structure. The objective of supervised network embedding is to improve models for a particular job, such graph or node classification, and to train embeddings for a specific purpose, like predicting graph or node properties. In Section 3, we go into further depth on the distinctions between supervised and unsupervised approaches, and we utilize the amount of supervision to construct our taxonomy.

A Taxonomy of Graph Embedding Models

We first describe our proposed framework, GraphEDM, a general framework for GRL (Sec- tion 3.1). In particular, GraphEDM is general enough that it can be used to succinctly de- scribe over thirty GRL methods (both unsupervised and supervised). We use GraphEDM to introduce a comprehensive taxonomy in Section 3.2 and Section 3.3, which summarizes exiting works with shared notations and simple block diagrams, making it easier to under- stand similarities and differences between GRL methods.

The GraphEDM framework

The GraphEDM framework builds on top of the work of Hamilton et al. (2017b), which describes unsupervised network embedding methods from an encoder-decoder perspective.

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY IJPAST
ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

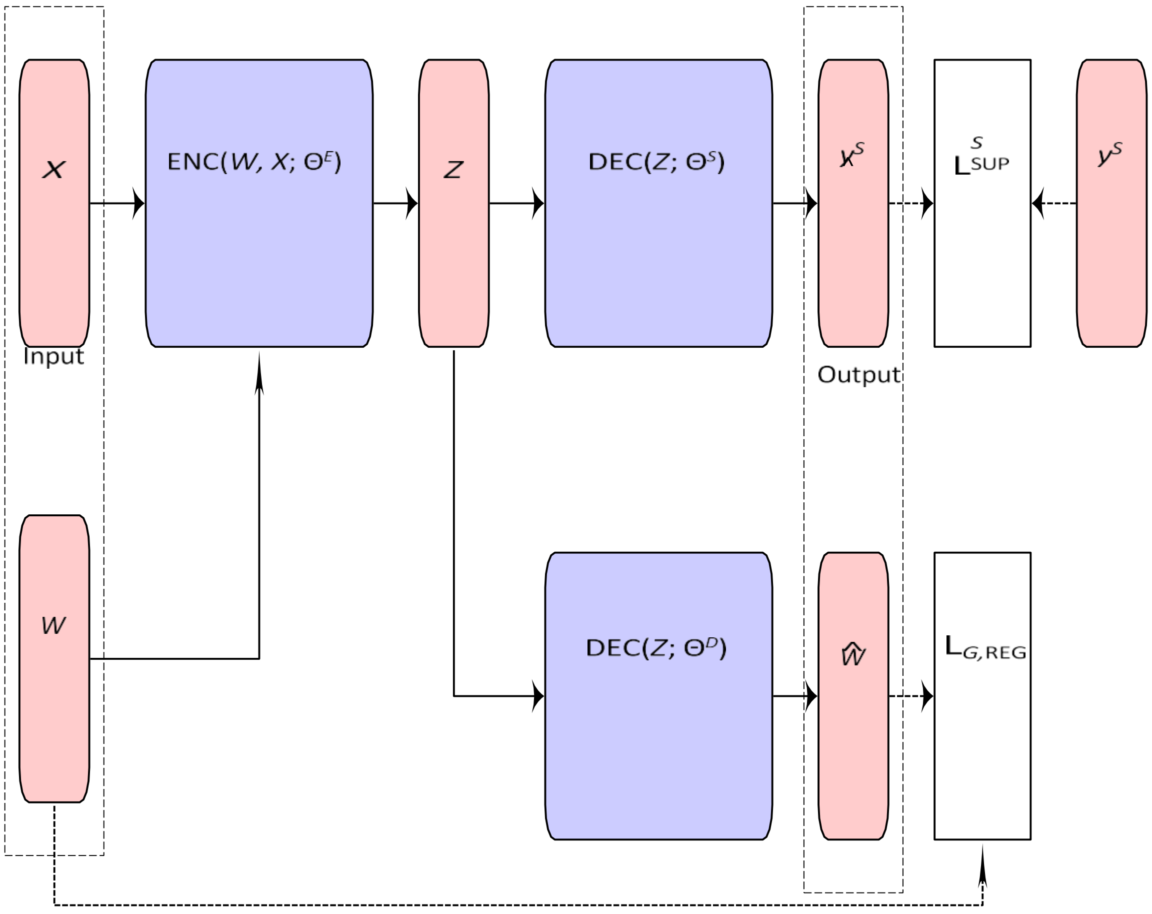


Figure 2: Illustration of the GRAPHEDM framework. Based on the supervision available, methods will use some or all of the branches. In particular, unsupervised methods do not leverage label decoding for training and only optimize the similarity or dissimilarity decoder (lower branch). On the other hand, semi-supervised and supervised methods leverage the additional supervision to learn models' parameters (upper branch).

Cruz et al. (2019) also recently proposed a modular encoder-based framework to describe and compare unsupervised graph embedding methods. Different from these unsupervised frameworks, we provide a more general framework which additionally encapsulates super- vised graph embedding methods, including ones utilizing the graph as a regularizer (e.g. Zhu and Ghahramani (2002))^E, and graph neural networks such as ones based on message passing (Gilmer et al., 2017; Scarselli et al., 2009) or graph convolutions (Bruna et al., 2014; Kipf and Welling, 2016a).

Input The GRAPHEDM framework takes as input an undirected weighted graph $G = (V, E)$, with adjacency matrix $W \in \mathbb{R}^{|V| \times |V|}$, and optional node features $X \in \mathbb{R}^{|V| \times d_0}$. In (semi-)supervised settings, we assume that we are given training target labels for nodes (denoted N), edges (denoted E), and/or for the entire graph (denoted G). We denote the supervision signal as $S \in \{N, E, G\}$, as presented below.

Model The GRAPHEDM framework can be decomposed as follows:

Graph encoder network $\mathrm{ENC}_{\Theta^E} : \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times d_0} \to \mathbb{R}^{|V| \times d}$, parameterized by Θ , which combines the graph structure with node features (or not) to produce node embedding matrix $Z \in \mathbb{R}^{|V| \times d}$ as:

$Z = \mathrm{ENC}(W, X; \Theta^E)$.

As we shall see next, this node embedding matrix might capture different graph prop- erties depending on the supervision used for training.

Graph decoder network $\mathrm{DEC}_{\Theta^D} : \mathbb{R}^{|V| \times d} \to \mathbb{R}^{|V| \times |V|}$, parameterized by Θ^D, which uses the node embeddings Z to compute similarity or dissimilarity scores for all no$y$d^{E}e pairs, producing a matrix $\widehat{W} \in$

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY IJPAST
ISSN 2229-6107 www.ijpast.in
Vol 9,Issue 1.Jan 2019

$R^{|V|\times|V|}$ as:
$\hat{W} = \mathrm{DEC}(Z; \Theta^D)$.
Classification network $\mathrm{DEC}_{\Theta^S} : R^{|V|\times d} \to R^{|V|\times|Y|}$, where Y is the label space. This network is used in (semi-)supervised settings and parameterized by Θ .
The output is a distribution over the labels $\hat{y}^S$, using node embeddings, as:
$\hat{y}^S = \mathrm{DEC}(Z; \Theta^S)$.

Our GRAPHEDM framework is general (see Fig. 2 for
an illustration). Specific choices of the aforementioned G (encoder and decoder) networks allows GRAPHEDM to
realize specific graph embedding methods. Before presenting the taxonomy and showing realizations of various methods using our framework, we briefly discuss an application perspective.

Output The GRAPHEDM model can return a reconstructed graph similarity or dissim-
ilarity matrix $\hat{W}$ (often used to train *unsupervised* embedding algorithms), as well as a
output labels $\hat{y}^S$ for *supervised* applications. The label output space Y varies depending on the supervised application.

Node-level supervision, with $\hat{y}^N \in Y^{|V|}$, where Y represents the node label space. If Y is categorical, then this is also known as (semi-)supervised node classification (Section 6.2.1), in which case the label decoder network produces labels for each node in the graph. If the embedding dimensions *d* is such that *d* =
|Y|, then the label decoder network can be just a simple softmax activation across the rows of *Z*, produc-
ing a distribution over labels for each node. Additionally, the graph decoder network might also be used in supervised node-classification tasks, as it can be used to regu- larize embeddings (e.g. neighbor nodes should have nearby embeddings, regardless of node
$\{\Theta^E, \Theta^D, \Theta^S\}$ denote all model parameters. ing a combination of the following loss terms:

Supervised loss term, L^S , which compares the labels).

Edge-level supervision, with $\hat{y}^E \in Y^{|V|\times|V|}$, where Y represents the edge label space. For example, Y can be
multinomial in knowledge graphs (for describing the
types of relationships between two entities), setting $Y = \{0, 1\}^{\#(\text{relation types})}$. It is
common to have #(relation types) = 1, and this is is known as *link* nomenclature and position link prediction as an *unsupervised* task (Section 4). Then in lieu of $\hat{y}^E$ we utilize $\hat{W}$, the output of the graph decoder network (which is learned to reconstruct a target similarity or dissimilarity matrix) to rank potential edges.

Graph-level supervision, with $\hat{y}^G \in Y$, where Y is the graph label space. In the graph classification task (Section 6.2.2), the label decoder network converts node embeddings into a single graph labels, using *graph pooling* via the graph edges captured by W . More concretely, the graph pooling operation is similar to pooling in standard CNNs, where the goal is to downsample local feature representations to capture higher-level information. However, unlike images, graphs don't have a regular grid structure and it is hard to define a pooling pattern which could be applied to every node in the graph. A possible way of doing so is via graph coarsening, which groups similar nodes into clusters to produce smaller graphs (Defferrard et al., 2016). There exist other pooling methods on graphs such as DiffPool (Ying et al., 2018b) or SortPooling (Zhang et al., 2018b) which creates an ordering of nodes based on their structural roles in the graph. Details about graph pooling operators is outside the scope of this work and we refer the reader to recent surveys (Wu et al., 2019) for a more in-depth treatment.
Taxonomy of objective functions
We now focus our attention on the optimization of models that can be described in the GRAPHEDM framework by describing the loss functions used for training. Let Θ =

Historical Context

There is a general two-step process that most machine learning models adhere to. Initially, they forego the need of human

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

feature building in favor of automatically extracting significant patterns from data. According to Bengio et al. (2013), this is the part where representation learning takes place. A second step involves putting these representations to use in supervised (like classification) or unsupervised (like clustering, visualization, and nearest-neighbor search) applications further down the line. This task is referred to as downstream processing.3 To facilitate the downstream process, a good data representation should be both expressive and concise, preserving the original data's significant qualities. Overfitting and other problems induced by the curse of dimensionality may be mitigated, for example, by using low-dimensional representations of high-dimensional datasets. When it comes to GRL, a graph encoder is used for representation learning, while a graph or label decoder is employed for jobs further down the line, such as node classification and link prediction. Graph encoder-decoder networks have traditionally been used for manifold learning. It is usual to presume that input data, even if it exists on a high-dimensional Euclidean space, is inherently contained on a low-dimensional manifold. The classic manifold hypothesis describes this. This inherently low-dimensional manifold is what manifold learning methods aim to retrieve. A discrete approximation of the manifold is often constructed initially, in the form of a graph with edges connecting adjacent points in the ambient Euclidean space. Graph distances are a reasonable surrogate for local and global manifold distances because manifolds are locally Euclidean. Secondly, while keeping graph distances as accurate as feasible, "flatten" this representation of the graph by learning a non-linear mapping from graph nodes to points in low-dimensional Euclidean space. Typically, these representations are more manageable compared to the initial high-dimensional ones, and they may subsequently be used in subsequent applications.

When looking for solutions to the manifold learning issue, non-linear4 dimensionality reduction strategies were all the rage in the early 2000s. For example, spectral approaches are used by Laplacian Eigenmaps (LE) (Belkin and Niyogi, 2002) to calculate embeddings, and IsoMap (Tenenbaum et al., 2000) to maintain global network geodesics by a mix of the Floyd-Warshall algorithm and the conventional Multi-dimensional scaling algorithm. In Section 4.1.1, we outline a few of these techniques that use shallow encoders. Despite their significant influence on machine learning, manifold dimensionality reduction approaches are not scalable to big datasets. Consider the time complexity of IsoMAP: it exceeds quadratic time due to the need to compute all pairs of shortest pathways. Since the mappings from node to embeddings are non-parametric, they cannot generate embeddings for additional datapoints, which is a potentially more significant drawback. The issue of graph embedding has seen several proposals for non-shallow network topologies in recent years. Our GRAPHEDM framework may be used to define graph neural networks and graph regularization networks. When compared to traditional approaches, GRL models often provide more expressive, scalable, and generalizable embeddings due to their use of deep neural networks' expressiveness.

In the next sections, we review recent methods for supervised and unsupervised graph embedding techniques using GRAPHEDM and summarize the proposed taxonomy in Fig. 3.

Unsupervised Graph Embedding

Using the taxonomy outlined earlier, we will now provide a summary of current methods for unsupervised graph embedding. Without using task-specific labels for the network or its nodes, these approaches map the graph into a continuous vector space, including its edges and/or nodes. By learning to rebuild matrices that measure the similarity or dissimilarity between nodes, such as the adjacency matrix, some of these approaches aim to learn embeddings that maintain the network structure. There are methods that use a contrastive objective. For example, one could compare nearby node-pairs to faraway ones: nodes that are co-visited in short random walks should have a higher similarity score than distant ones. Another would compare real graphs to fake ones: the mutual information between a graph and all of its nodes should be higher in real graphs than in fake ones.

Shallow embedding methods

The encoder function in shallow embedding techniques is a basic embedding lookup; these methods are transductive graph embedding methods. The shallow encoder function is simply: for every node vi in V, there is a corresponding low-dimensional learnable embedding vector Zi in Rd.

$$Z = \mathrm{ENC}(\Theta^E) = \Theta^E \in \mathbb{R}^{|V|\times d}.$$

The data structure in the embedding space matches the underlying graph structure, thanks to learnt node embeddings. Generally speaking, it's not dissimilar to principal component analysis (PCA) and other dimensionality reduction techniques; however, the input data may not be linear. Specifically, graph embedding issues may be addressed using techniques for non-linear dimensionality reduction, which often begin with constructing a discrete graph from the data in order to approximate the manifold. We take a look at the distance-based and outer product-based approaches to shallow graph embedding.

Distance-based methods By using a preset distance function, these approaches maximize embeddings in a way that keeps points that are close together in the graph (as shown by their graph distances, for example) as near together in the embedding space as feasible. In a formal sense, the decoder network may provide either non-Euclidean (Section 4.1.2) or Euclidean (Section 4.1.1) embeddings by computing pairwise distance for a certain distance function d2:

We now cover spectrum-free methods, which approximate convolutions in the spectral do- main overcoming computational limitations of SCNNs by avoiding

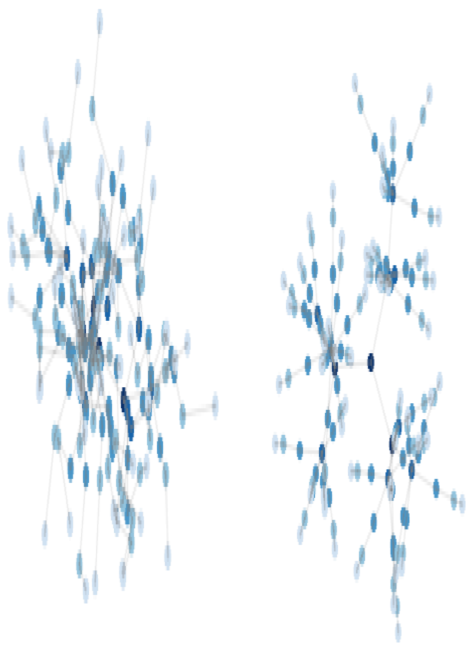
ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

explicit
computation of
the Laplacian’s
eigendecompositi
on.
SCNNs filters

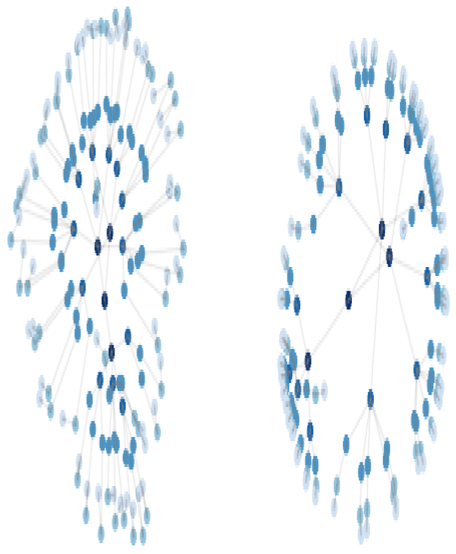
are neither
localized nor
paramet-
in Eq. (17) are all free. To overcome
this

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

issue, sprectrum-free methods use polynomial expansions to approximate



(a) GCN layers.



(b) HGCN layers.

Figure 13: Euclidean (left) and hyperbolic (right) embeddings of a tree graph. Hyperbolic embeddings learn natural hierarchies in the embedding space (depth indicated by color). Reprinted with permission from (Chami et al., 2019).

Non-Euclidean Graph Convolutions

Hyperbolic shallow embeddings enable embeddings of hierarchical graphs with smaller dis- tortion than Euclidean embeddings. However, one major downside of shallow embeddings is that they are inherently transductive and cannot generalize to new graphs. On the other hand, Graph Neural Networks, which leverage node features, have achieved state-of-the-art performance on inductive graph embedding tasks.Recently, there has been interest in extending Graph Neural Networks to learn non-Euclidean embeddings and thus benefit from both the expressiveness of Graph Neural Networks and hyperbolic geometry. One major challenge in doing so is how to perform convolutions in a non- Euclidean space, where standard operations such as inner products and matrix multiplications are not defined.

Hyperbolic Graph Convolutional Neural Networks (HGCN) (Chami et al., 2019) and Hyperbolic Graph Neural Networks (HGNN) (Liu et al., 2019) apply graph convolutions in hyperbolic space by leveraging the Euclidean tangent space, which provides a first-order approximation of the hyperbolic manifold at a point. For every graph convolution step, node embeddings are mapped to the Euclidean tangent space at the origin, where convolutions are applied, and then mapped back to the hyperbolic space. These approaches yield significant improvements on graphs that exhibit hierarchical structure (Fig. 13).

Summary of supervized graph embedding

This section presented a number of methods that process task labels (e.g., node or graph labels) at training time. As such, model parameters are directly optimized on the upstream task.

Shallow methods use neither node features X nor adjacency W in the encoder (Section 5.1), but utilize the adjacency to ensure consistency. Such methods are useful in transductive settings, if only one graph is given, without node features, a fraction of nodes are

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

labeled, and the goal is to recover labels for unlabeled nodes.

Applications

Many different kinds of applications, both supervised and unsupervised, may benefit from graph representation learning techniques. When learning embeddings in an unsupervised setting, task-specific labels are not processed. Instead, the graph serves as a tool for self- monitoring. Using unsupervised embedding techniques (Section 4, top branch of the Taxonomy in Fig. 3), one may learn embeddings that preserve the network (i.e. neighborhoods) or the structural equivalence of nodes (for distinction, see Section 2.2.3). Alternatively, in supervised applications, such as graph or node classification, the optimization of node embeddings is done directly for a particular job. Section 5, the bottom branch of the Taxonomy in Figure 3, describes supervised embedding approaches that may be used in this context. Here are a few of the most common GRL jobs and the methods used to do them, as shown in Table 5. What follows is a rundown of typical

supervised and unsupervised graph uses.

Unsupervised applications

GRAPH RECONSTRUCTION

Graph reconstruction is the gold standard for unsupervised graph applications. The objective here is to train mapping functions (parametric or not) that retain graph features like node similarity while mapping nodes to dense distributed representations. By reducing a reconstruction error—the error in retrieving the original graph from learnt embeddings—models may be trained, and graph reconstruction doesn't need any supervision. For some instances of reconstruction aims, see Section 4, and to learn about the techniques used for this purpose, see Section 5. Similar to dimensionality reduction, the overarching objective of graph reconstruction is to combine incoming data into a low-dimensional representation. Graph reconstruction models aim to compress data specified on graphs into low-dimensional vectors, rather than the usual way of reducing dimensionality (e.g., principal component analysis) which involves converting high-dimensional vectors into low- dimensional ones.

10.1.1 LINK PREDICTION

The goal of link prediction is to forecast which edges in a graph will eventually take a certain path. To rephrase, link prediction tasks aim to anticipate the appearance of linkages that have not yet been detected, such as links that might emerge in the future for networks that are both dynamic and temporal. Furthermore, malicious links may be located and eliminated with the use of link prediction. Common examples of this kind of application are recommendation systems that utilize graph learning models to forecast the interactions between users and products and social networks that use these models to forecast the friendships between users.

	Method	**Training complexity**		**Training input**
		Memory	Computation	
(a)	DeepWalk (Perozzi, 2014)	$O(\|V\|d)$	$O(c^2d\|V\|\log_2\|V\|)$	
(b)	node2vec (Grover, 2016)	$O(\|V\|d)$	$O(c^2d\|V\|)$	

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

(c)	LINE (Tang, 2015) HOPE (Ou, 2016) GF (Ahmed, 2013)	$O(\|V\|d)$	$O(\|E\|d)$	W
(d)	SDNE (Wang, 2016) DNGR (Cao, 2016)	$O(\|V\|b\mathrm{D})$	$O(\|V\|b\mathrm{M})$	
(e)	GraRep (Cao, 2015) WYS (Abu-el-haija, 2018)	$O(\|V\|^2)$	$O(\|V\|^3c + \|V\|^2d)$	
(f)	HARP (Chen, 2018)	*inherits*		W
(g)	Splitter (Epasto, 2019)	*inherits*		W
(h)	MDS (Kruskal, 1964)	$O(\|V\|^2)$	$O(\|V\|^3)$	X induces W
(i)	LP (Zhu, 2002) LLE (Roweis, 2000)	$O(\|V\|)$	$O(\|E\| \times \text{iters})$	
(j)	GNN Methods	$O(\|V\|\mathrm{D})$	$O(\|E\|\mathrm{D} + \|V\|\mathrm{M})$	X, W
(k)	SAGE (Hamilton, 2017)	$O(bF^H\mathrm{D})$	$O(bF^{H-1}\mathrm{D} + bF^H\mathrm{M})$	X, W
(l)	GTTF (Markowitz, 2021)	$O(bF^H\mathrm{D})$	$O(bF^{H-1}\mathrm{D} + bF^H\mathrm{M})$	X, W

ISSN 2229-6107 www.ijpast.in
Vol 9,Issue 1.Jan 2019

Summarization and real-world applications of GRL techniques (Table 5). The columns running from right to left show the following: method classes, the hardware cost to train the method, and real cases where the methods have been useful: inputs to the methods, which may be either an adjacency matrix (W) or node characteristics (X), or both. This is how we get the Training Complexity. In the method classes (a-h), "c" represents the size of the context (such as the length of a random walk) and "d" the size of the embedding dictionary; both are parameters of node embedding techniques. The embedding dictionary is stored in (a) DeepWalk and (b) node2vec, with (V d) floating-point entries. During training, a predetermined number of walks with a defined duration are simulated from every node V. Along these walks, the dot products of all node-pairs within a window of size c are computed. Both the hierarchical softmax (a) and the negative sampling (b) are applied to every pair. To see the complexity per batch, just replace the two V terms on the left with batch size b. But to keep things simple, we look at it per period. (c) All edges are cycled through by LINE (Tang, 2015), HOPE (Ou, 2016), and GF (Ahmed, 2013). (d) The adjacency matrix is used to train auto- encoders via SDNE and DNGR, with batch- size b, and the total dimensions of all layers denoted by A dA. To handle floating-point

operations in matrix multiplications, the formula = A dAdA+1 is used. With full-batch, b equals V. (e) GraRep and WYS store a dense square matrix with (V 2) non-zero elements, and they elevate the transition matrix to the power of c. Their complexity is algorithm- specific since (f) HARP (Chen, 2018) and (g) Splitter can execute any algorithm, for example, (a-e). In this case, we assume that both the average number of persons per node for Splitter and the number of times HARP is activated (the graph's scales) are minimal (V).

(h) While LE necessitates the entire eigendecomposition of the graph laplacian matrix (to get the eigenvectors corresponding to the fewest eigenvalues), MDS calculates all-pairs similarity. If the number of label classes is small, (i) LP and LLE will loop over edges up to "iters" iterations. (j) include GCN, GAT, MixHop, GIN, GGNN, MPNN, ChebNet, and MoNet graph convolution algorithms (Kipf, 2016; Defferrard, 2016;

Abu-el-haija, 2019; Xu, 2018; Li, 2015; Gilmer, 2017; Xu, 2018; Xu, 2018; Monti, 2017). The creators of those techniques gave a full-batch implementation, which we presume is naïve. After adding up all of the floating- point operations performed by its neighbors (a total of E floats), each node in a given layer multiplies that total by the layer filter (a total of V floats). Lastly, sampling approaches such as (k-l) enable learning to scale to bigger networks by reducing the hardware required of the training algorithm and separating memory complexity from graph size. (k) For each node in the batch (with a size of b), (l) GTTF samples F nodes, and for each node's neighbors, F as well. This continues until the tree height reaches H. We disregard the runtime complexity of data pre-processing for(k) and (l) since it has to be calculated only once per graph, independent of the number of (hyperparameter) sweep computations. A common approach for training link prediction models is to mask some edges in the graph (positive and negative edges), train a model with the remaining edges and then test it on the masked set of edges. Note that link prediction is different from graph reconstruction.

ISSN 2229-6107 www.ijpast.in
Vol 9,Issuse 1.Jan 2019

In link prediction, we aim at predicting links that are not observed in the original graph while in graph reconstruction, we only want to compute embeddings that preserve the graph structure through reconstruction error minimization.

Finally, while link prediction has similarities with supervised tasks in the sense that we have labels for edges (positive, negative, unobserved), we group it under the unsupervised class of applications since edge labels are usually not used during training, but only used to measure the predictive quality of embeddings. That is, models described in Section 4 can be applied to the link prediction problem.

12.1.1 CLUSTERING

3. The discovery of communities is one of the numerous real-world applications of clustering. For example, clusters may be seen in biological networks (as collections of proteins with shared characteristics) or social networks (as associations of individuals with same interests).

Keep in mind that clustering issues may be solved using the unsupervised approac hes discussed in this review. For example, one might apply a clustering algorithm, such as k- means, to embeddings that are produced by an encoder. Another option is to include clustering into the learning process while using a shallow or Graph Convolution embedding model (Rozemberczki et al., 2019; Chiang et al., 2019; Chen et al., 2019a).

13.1.1 VISUALIZATION

4. For visualizing graphs, there are several ready-made tools that map nodes onto two- dimensional manifolds. Network scientists are able to get a qualitative understanding of graph characteristics, node interactions, and node clusters via the use of visualizations. Force-Directed Layouts-based approaches with different web- app Javascript implementations are among the popular tools. To achieve this visualization, one can use an unsupervised graph embedding method such as t- distributed stochastic neighbor embeddings (t-SNE) or principal component analysis (PCA) after training an encoder-decoder model (which is equivalent to a shallow embedding or graph convolution network) (Maaten and Hinton, 2008; Jolliffe, 2011). Graph learning techniques are often evaluated qualitatively using this approach (embedding → dimensionality reduction). To color the nodes in 2D visualization plots, one may utilize their characteristics if the nodes have any. As seen in visual representations of different approaches, good embedding algorithms place nodes in the embedding space that have comparable properties close together (Perozzi et al., 2014; Kipf and Welling, 2016a; Abu-El-Haija et al., 2018). To conclude, approaches that map every graph to a representation may also be projected into two dimensions to display and qualitatively assess graph-level features, in addition to mapping every node to a 2D coordinate (Al-Rfou et al., 2019).

4.1 Supervised applications

4.1.1 Node classification

5. An essential supervised graph application is node classification, which aims to develop representations of nodes that can reliably predict their labels. In citation networks, node labels may represent scientific subjects; in social networks, they might represent gender and other characteristics. One typical use case is semi-supervised node classification due to the high cost and time commitment associated with labeling huge graphs. The objective in semi-supervised situations is to use node linkages to predict characteristics of unlabeled nodes, with just a small percentage of nodes being tagged. Since there is a single partly labeled fixed graph in this context, it is considered transductive. Inductive node classification is another option; this is the process of determining how to categorize nodes in different networ ks. Keep in mind that if the node attributes are descriptive of the goal label, they may greatly improve performance on classified nodes jobs. In fact, by integrating structural data with semantic information derived from features, state-of-the-art performance on multiple node classification benchmarks has been attained by more recent approaches as GCN (Kipf and Welling, 2016a) or GraphSAGE (Hamilton et al., 2017a). However, other approaches, such random walks on graphs, do not take use of feature information and so perform worse on these tasks.

15.1.1 Graph classification

One example of a supervised application is graph classification, the goal of which is to use an input graph to predict labels at the graph level. Due to the constant introduction of novel graphs during testing, graph classification problems are fundamentally inductive. Biochemical activities and online social networks are also common choices. Graphs representing molecules are often used in the biological field. A feature vector that is a 1-hot encoding of an atom's number may serve as a node in these graphs, and a bond can be represented by an edge between two nodes, with the kind of the bond being indicated by the feature vector. One example of a task-dependent graph-level label is MUTANG, which indicates the mutagenicity of a medicine against bacteria (Debnath et al., 1991). Typically, people are represented as nodes in online social networks, while connections or interactions are symbolized by edges. As an example, there are a lot of graphs in the Reddit graph classification jobs (Yanardag and Vishwanathan, 2015). An edge will link two nodes in a graph that represents a conversation thread, such as when one person comments on another's remark.

Given a comment graph, the objective is to identify the community (sub-reddit) where the conversation occurred. While tasks such as node classification and edge prediction include pooling at the node and edge levels, respectively, graph classification tasks need a different kind of pooling to aggregate data at the node and graph levels. As said before, expanding this concept of pooling to any kind of graph is a challenging and ongoing topic of study. Node order shouldn't affect the pooling function. For example, several approaches use basic pooling, including taking the mean or total of all latent vectors at the node level in the network (Xu et al., 2018). Ying et al., 2018b; Cangea et al., 2018; Gao and Ji, 2019; Lee et al., 2019 are among the approaches that employ differentiable pooling. Tsitsulin et al. (2018a), Al-Rfou et al. (2019), and Tsitsulin et al. (2020a) all provide supervised approaches for learning graph- level representations, but there are also many unsupervised methods. Some unsupervised graph-level models that stand out include reviewed by Viswanathan et al. (2010) and Kriege et al. (2020) as graph kernels (GKs).Although GKs are not our primary concern, we do touch on their links to GRAPHEDM here. Graph-level tasks, such graph categorization, are suitable for GKs. In order to convert any two graphs into a scalar, GK may automatically apply a similarity function. Counting the number of walks (or pathways) that two graphs have in common is one way that traditional GKs calculate graph similarity. For example, each walk may be stored as a series of node labels. Common practice dictates using node degrees as labels in the absence of explicit labels. The capacity of GKs to identify (sub-)graph isomorphism is a common metric for analysis. When ordering of nodes is ignored, two (sub-)graphs are considered isomorphic if they are identical. According to the 1-dimensional Weisfeiler-Leman (1-WL) heuristic, two sub-graphs are considered isomorphic since sub-graph isomorphism is NP-hard. In each graph, histograms are used to tally the statistics of the nodes (e.g., how many nodes with the label "A" have an edge to nodes with the label "B"). If two graphs' histograms, obtained from the same 1-hop neighborhood, are equal, then the graphs are considered isomorphic according to the 1-WL heuristic. An example of a GNN that has been shown to achieve the 1-WL heuristic is the Graph Isomorphism Network (GIN; Xu et al., 2018). This means that GIN can only map two graphs to the same latent vector if they are considered isomorphic according to the 1-WL heuristic. In some newer studies, GKs and GNNs are used together. Using the similarity of the "tangent space" of the goal with respect to the Gaussian-initialized GNN parameters, Du et al. (2019) models the similarity of two graphs, and Chen et al.

(2020) extracts walk patterns. There isn't any GNN training in either (Du et al., 2019; Chen et al., 2020). Instead, kernel support vector machines and other kernelized algorithms are used to the pairwise Gram matrix during training. Therefore, our GCF and GRAPHEDM frameworks are not well-suited to include these methodologies. However, there are other approaches that don't rely on indirectly computing graph-to-graph similarity scalar scores but instead directly map graphs to high-dimensional latent spaces. One example is Morris et al.'s (2019) k-GNN network, which is deliberately coded as a GNN but can actually implement the k-WL heuristic (which is identical to 1-WL but where histograms are produced up-to k-hop neighbors). Therefore, our GCF and GRAPHEDM frameworks can define the k-GNN model classes.

Conclusion

Open Research Directions We presented a standard method for comparing ML models trained on graph-structured data in this survey. Deep graph embedding techniques, graph auto-encoders, graph regularization techniques, and graph neural networks are all included in our expanded GRAPHEDM framework, which was before used for unsupervised network embedding. Additionally, we presented a graph convolution framework (GCF) for describing and comparing graph neural networks that rely on convolution, such as spatial and spectral graph convolutions in particular. We included more than 30 supervised and unsupervised techniques for graph embedding in our exhaustive taxonomy of GRL methods, which we presented using this framework.
With any luck, the results of this poll will inspire further

GRL research, which should lead to solutions for the problems these models are experiencing right now. The taxonomy is very useful for practitioners since it helps them understand the many tools and applications available and makes it easy to choose the right technique for each situation. Furthermore, academics who have just published Researchers may use the taxonomy to organizetheir inquiries, locate relevant literature, establish reliable baselines for comparison, and choose suitable methods for data analysis.

Although GRL approaches have shown to be very effective in node classification and link prediction, there are still several issues that need to be addressed. We then go on to talk about the difficulties and future prospects of graph embedding models in terms of research.

Evaluation and benchmarks

Standard benchmarks for node classification or link prediction are usually used to evaluate the approaches presented in this review. To illustrate the point, graph embedding techniques are often evaluated against citation networks. The findings may differ greatly depending on the datasets' splits or training processes (such as early halting), which is a problem with these tiny citation benchmarks, as shown in recent research (Shchur et al., 2018).

Using strong and consistent evaluation methodologies, as well as expanding the scope of assessment beyond small node categorization and link prediction benchmarks, is crucial for the improvement of GRL approaches. New graph benchmarks with leaderboards (Hu et al., 2020; Dwivedi et al., 2020) and graph embedding libraries (Fey and Lenssen, 2019; Wang et al., 2019; Goyal and Ferrara, 2018a) are examples of recent development in this approach. Similarly, in order to test GNNs' reasoning skills, Sinha et al. (2020) suggested a series of exercises based on first- order logic.

Fairness in Graph Learning To prevent models from correlating'sensitive' characteristics with the model's predicted output, a new area called Fairness in Machine Learning is developing (Mehrabi et al., 2019). Considering the association of the graph structure (the edges) and the feature vectors of the nodes with the final output, these considerations might be particularly significant for graph learning challenges.

Bose and Hamilton (2019) state that adversarial learning is the most prevalent method for implementing fairness requirements in models. This method may be used to GRL in order to debias the model's predictions with respect to the sensitive feature(s). But there are no certain assurances on the precise amount of bias eliminated using adversarial approaches. The debiasing job itself may be difficult to accomplish with several debiasing strategies (Gonen and Goldberg, 2019). Provable guarantees for debiasing GRL have been the focus of recent work in the field (Palowitch and Perozzi, 2019).

Application to large and realistic graphs

Graph learning techniques are typically reserved for datasets of tens of thousands to hundreds of thousands of nodes. Still, there are far bigger graphs in the actual world, with billions of nodes. A Distributed Systems configuration with several computers, like MapReduce, is necessary for methods that scale for big graphs (Lerer et al., 2019; Ying et al., 2018a) (Dean and Ghemawat, 2008). Is there a way for a researcher to use a home computer to apply a learning approach to a very big graph that fits on a single hard drive (e.g., with a one terabyte size) but does not fit on RAM? See how this stacks up against a computer vision challenge using a large picture collection (Deng et al., 2009; Kuznetsova et al., 2020). Any model that can fit on RAM can be trained on personal computers, regardless of the size of the dataset. Graph embedding models, in particular those whose parameters grow in size as the graph's nodes do, may find this issue very difficult to solve.

Even picking the right graph to utilize as input might be challenging at times in business. The

ISSN 2229-6107 www.ijpast.in
Vol 9,Issue 1.Jan 2019

Google system Grale, which learns the right graph from several characteristics, is described by Halcrow et al. (2020). For graph learning on massive datasets, Grale uses similarity search methods (such as locality sensitive hashing). A recent study by Rozemberczki et al. (2021) adds an attention network to the Grale model, enabling end-to-end learning.We anticipate that learning algorithms for big graphs that are still executable on a single computer will present new mathematical and practical problems. We are hopeful that scholars would prioritize this area so that non-expert practitioners, like a neurology researcher, may access and use these learning methods to evaluate the human brain's sub-graph, which is comprised of neurons and synapses represented as nodes and edges.

Molecule generation Learning on graphs has a great potential for helping molecular scientists to reduce cost and time in the laboratory. Researchers proposed methods for predicting quantum properties of molecules (Gilmer et al., 2017; Duvenaud et al., 2015) and for generating molecules with some desired properties (Liu et al., 2018; De Cao and Kipf, 2018; Li et al., 2018; Simonovsky and Komodakis, 2018; You et al., 2018). A review of recent methods can be found in (Elton et al., 2019). Many of these methods are concerned with manufacturing materials with certain properties (e.g. conductance and malleability), and others are concerned drug design (Jin et al., 2018; Ragoza et al., 2017; Feng et al., 2018).

Combinatorial optimization

Numerous fields encounter computationally challenging challenges, such as routing science, cryptography, decision-making, and planning. Computationally hard problems are those for which the techniques used to find the best solution have poor scalability. We cite (Bengio et al., 2018) for a summary of the ways that have recently attracted attention insolving combinatorial optimization issues by using machine learning approaches, such as reinforcement learning.

Recently, there has been interest in using graph embeddings to approximate solutions to NP-hard problems (Khalil et al., 2017; Nowak et al., 2017; Selsam et al., 2018; Prates et al., 2019). Graphs are a natural representation for many hard issues, such as SAT and vertex cover; in fact, many problems may be described in terms of graphs. These techniques use a data-driven approach to solving computationally difficult issues, such as determining whether a specific instance (e.g., node) is part of the best solution from among many instances of the problem. Find assignments that strive to accomplish a goal (e.g., the minimal conductance cut) in other works that optimize graph partitions (Bianchi et al., 2020; Tsitsulin et al., 2020b). All of these methods use GNNs as their starting point since GNNs, thanks to their relational inductive biases, can better depict graphs than regular neural networks (e.g. permutation invariance). Current solutions still outperform these data-driven approaches, but GNNs have shown promise in generalizing to bigger problem cases (Nowak et al., 2017; Prates et al., 2019). Lamb et al. (2020) provides a comprehensive overview of GNN- based approaches to combinatorial optimization in their latest study on neural symbolic learni ng.

ISSN 2229-6107 www.ijpast.in
Vol 9,Issue 1.Jan 2019

Non-Euclidean embeddings The underlying space geometry is an important part of graph embeddings, as we saw in Sections 4.1.2 and 5.6. All graphs are discrete complex, non-Euclidean structures with high dimensions; however, there is currently no simple method for encoding such data into embeddings with low dimensions that maintain the graph topology (Bronstein et al., 2017). Hyperbolic and mixed- product space embeddings are two examples of non- Euclidean embeddings that have recently attracted attention and made strides in the field of learning (Gu et al., 2018; Nickel and Kiela, 2017). In comparison to their Euclidean counterparts, these non-Euclidean embeddings have the potential for embeddings that are more expressive. For example, compared to Euclidean embeddings, hyperbolic embeddings exhibit significantly less distortion when representing hierarchical data (Sarkar, 2011). This has led to state- of-the-art outcomes in numerous contemporary applications, including linguistics tasks (Tifrea et al., 2018; Le et al., 2019) and knowledge graph link prediction (Balazevic et al., 2019; Chami et al., 2020).

Non-Euclidean embeddings often bring two difficulties: first, hyperbolic space precision problems (e.g., at the Poincar'e ball boundary) (Sala et al., 2018; Yu and De Sa, 2019), and second, difficult Riemannian optimization (Bonnabel, 2013; Becigneul and Ganea, 2018). Furthermore, it is not apparent how to choose the appropriate shape for an input graph. An intriguing area for future research is the process of selecting or learning the appropriate geometry for a specific discrete graph, even though there are already discrete measures for the graphs' tree-likeliness, such as Gromov's four-point condition (Jonckheere et al., 2008; Abu-Ata and Dragan, 2016; Chen et al., 2013; Adcock et al., 2013).

Assurances based on theory Recent developments in graph embedding model design have outperformed state-of-the- art methods in several domains. Nevertheless, our knowledge of the theoretical promises and constraints of graph embedding models is currently restricted. Xu et al. (2018), Verma and Zhang (2019), Morris et al. (2019), and Garg et al. (2020) all apply current findings from learning theory to the issue of GRL, which is a new field of study on GNN representational power. If we want to know what the theoretical benefits and drawbacks of graph embedding techniques are, we need to build theoretical frameworks.

References

networks: a research investigation. In: Networks, 2016; 67(1): 49–68.

Brian Perozzi, Sami Abu-El-Haija, and Rami Al-Rfou. Improving edge representations via low-rank asymmetric projections. Page 1787– 1796 of the 2017 ACM Conference on Information and Knowledge Management (CIKM '17) proceedings.With contributions from Bryan Perozzi, Alexander A. Alemi, Sami Abu-El-Haija, and Rami Al-Rfou. Be cautious: Finding node embeddings via observing graphs. Page numbers 9180–9190 from the 2018 edition of Advances in Neural Information Processing Systems. Participants: Aram Galstyan, Bryan Perozzi, Bryan Harutyunyan, Nazanin

By Feodor F. Dragan and Muad Abu-Ata. Structures resembling metric trees in actual

Alipourfard, Kristina Lerman, Greg Ver Steeg, and Amol Kapoor. Mixhop is a method for building higher-order graph convolutional networks by combining sparse neighborhoods. Page numbers 21–29 from the 2019 International Conference on Machine

Learni

ng.

Blair D. Sullivan, Michael W. Mahoney, and Aaron B. Adcock. Big social and information networks have a tree-like structure. Volume 13, Issue 1, Pages 1–10, 2013 IEEE

International Conference on Data Mining. 2013, IEEE. Vanja Josifovski, Alexan-der J. Smola, Nino Shervashidze, Shravan Narayanamurthy, and Amr Ahmed

composed the team. Organic graph factorization on a distributed, massive scale. Included in the proceedings of the 22nd international conference on the World Wide Web, pages 37-
48. IEEE, 2013. Everyone from Rami Al-Rfou to Dustin Zelle and Bryan Perozzi were involved. Ddgk: Deep divergence graph kernels represented by learned graphs. W3C 2019 Conference Proceedings on the World Wide Web, 2019.
By Miguel A. Andrade-Navarro, Pablo Mier, and Gregorio Alanis-Lobato. Efficiently incorporating complicated networks into hyperbolic space using their Laplacian? Submitted to the journal Scientific Reports on 2016-03-08.Almeida, Luis B. A com- binatorial learning rule for asynchronous perceptrons with feedback. Volume 2, pages 609-618, Proceedings of the First International Conference on Neural Networks. 1987, IEEE. Alan Allen, Ivana Balazevic, and Timothy Hospedales... "Multi-relational Poincaré graph embeddings" Pages 4463–4473 of the 2019 edition of Advances in Neural Information Processing Systems.Matt Lai, Danilo Jimenez Rezende, Peter Battaglia, Razvan Pascanu, and others. Connected systems for acquiring knowledge of physical phenomena, relationships, and objects. Page numbers 4502-4510 from the 2016 edition of Advances in Neural Information Processing Systems. Regarding relational inductive biases, deep learning, and graph networks, the following authors are involved: Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, Raposo, Adam Santoro, Ryan Faulkner, and others. 2018 arXiv preprint arXiv:1806.01261, published here.