Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Mathematics and Data Analysis

Empirical Validations of Graph Structure Learning methods
for Citation Network Applications

Hoang Thien Ly

student record book number 308933

main thesis supervisor

prof. dr hab. inż. Przemysław Biecek

assistant supervisor

mgr Duy Nguyen H.M

WARSAW 2023

Abstract

EMPIRICAL VALIDATIONS OF GRAPH STRUCTURE LEARNING METHODS FOR CITATION NETWORK APPLICATIONS

This Bachelor's Thesis aims to examine the classification accuracy of graph structure learning methods in graph neural networks domain, with a focus on classifying a paper in citation network datasets. Graph neural networks (GNNs) have recently emerged as a powerful machine learning concept allowing to generalize successful deep neural network to non-Euclidean structured data with high performance. However, one of the limitations of the majority of current GNNs is the assumption that the underlying graph is known and fixed. In practice, real-world graphs are often noisy and incomplete or might even be completely unknown. In such cases, it would be helpful to infer the graph structure directly from the data. Additionally, graph structure learning permits learning latent structures, which may increase the understanding of GNN models by providing edge weights among entities in the graph structure, allowing modellers to further analysis.

As part of the work, we will:

- review the current state-of-the-art graph structure learning (GSL) methods.
- empirically validate GSL methods by accuracy scores with citation network datasets.
- analyze the mechanism of these approaches and analyze the influence of hyperparameters on model's behavior.
- discuss future work.

This thesis presents a key finding that in dealing with graph-based datasets, graph-based neural networks outperform traditional neural networks (Multilayer Perceptron). Through an evolutionary progression of GSL techniques, starting from Graph Convolutional Networks (GCN) to Graph Attention Networks (GAT) and finally to Discrete DGM (dDGM), the validity and effectiveness of these approaches improve incrementally. This evolutionary process represents a paradigm shift in handling graph data, leading to enhanced performance showcased by validated results.

Keywords: graph neural network, graph structure learning, empirical validations, citation network applications.

Streszczenie

EMPIRYCZNA WALIDACJA METOD UCZENIA SIĘ STRUKTURY GRAFÓW DO ANALIZY SIECI CYTOWAŃ

Niniejsza praca licencjacka ma na celu zbadanie dokładności klasyfikacji metod uczenia się struktury grafów w domenie grafowych sieci neuronowych, ze szczególnym uwzględnieniem klasyfikacji artykułów w zbiorach danych sieci cytowań. Grafowe sieci neuronowe (GNN) pojawiły się niedawno jako potężna koncepcja uczenia maszynowego, pozwalająca na uogólnienie udanych głębokich architektur neuronowych na dane o strukturze nieeuklidesowej z wysoką wydajnością. Jednak jednym z ograniczeń większości obecnych GNN jest założenie, że bazowy graf jest znany i stały. W praktyce rzeczywiste grafy są często zaszumione i niekompletne, a nawet mogą być całkowicie nieznane. W takich przypadkach pomocne byłoby wnioskowanie o strukturze grafu bezpośrednio z danych. Dodatkowo, uczenie się struktury grafu pozwala na uczenie się ukrytych struktur, co może zwiększyć zrozumienie modeli GNN poprzez dostarczanie wag krawędzi między jednostkami w strukturze grafu, umożliwiając modelarzom dalszą analizę.

W ramach tej pracy dokonamy:

- dokonamy przeglądu aktualnych metod uczenia się struktury grafów (GSL).
- empirycznie zweryfikować metody GSL za pomocą wyników dokładności z zestawami danych sieci cytowań.
- przeanalizujemy mechanizm tych podejść i przeanalizujemy wpływ hyperparametrów na zachowanie modelu.
- omówienie przyszłych prac.

Niniejsza praca licencjacka przedstawia kluczowe odkrycie, że w przypadku zbiorów danych opartych na grafach, sieci neuronowe oparte na grafach przewyższają tradycyjne sieci neuronowe (Perceptron Wielowarstwowy). Poprzez ewolucyjny postęp technik GSL, począwszy od grafowych sieci konwolucyjnych (GCN), poprzez grafowe sieci uwagi (GAT), a skończywszy na dyskretnym DGM (dDGM), ważność i skuteczność tych podejść poprawia się stopniowo. Ten ewolucyjny proces stanowi zmianę paradygmatu w przetwarzaniu danych grafowych, prowadząc do zwiększonej wydajności popartej zweryfikowanymi wynikami.
Słowa kluczowe: grafowe sieci neuronowe, uczenie się struktury grafów, walidacje empiryczne, aplikacje sieci cytowań.

Contents

1. Introduction

A graph is a non-linear data structure that consists of nodes and edges joining nodes. This data structure is widely used in various fields due to the strength of representing entities and their complicated interactions. Some examples can be found in:

- **Chemistry** [1]: Molecules in general are the building blocks of atoms and electrons. Graph structure (specifically used: constitutional graph) has vertices symbolizing atoms and edges representing covalent bonds.
- **Transportation networks** [2]: Graph structures are used to represent transportation networks like road networks, subway systems, and airline routes. Nodes can represent cities or stations, and edges can represent distance between cities or stations.
- **Biology** [3]: in the protein-protein interaction network (PPIN) application of graph theory, the vertices represent proteins and the edges represent physical interactions between proteins. Edges in PPIN applications are typically directional.
- **Social networks** [4] : Social networks are tools in social science to study the patterns in collective relation of people or their behaviour in organizations. Graph structure can be used to represent social networks like Facebook, LinkedIn, Twitter. Nodes can represent entities (individuals or groups), and edges can represent relationships between them.
- **Internet**: Graph structures are used to represent the internet. A web can be treated as a graph vertex and each hyperlink as a graph edge.
- **Recommendation Systems** [5, 6]: Graph structures can be used to create recommendation systems that suggest products, movies, or books to users based on their preferences and behaviors.
- **Citation networks**: Scientists cite other scientists's paper. These citations can be visualized as a graph, where each paper is a node, and each directed edge is a citation among papers. In addition, we can add feature vector for each node to make the graph more informative [7].

Given the prevalence of graph-structured data in various fields, it is crucial to develop effective methods for learning the structure of graphs and utilizing this information to make predictions. Graph Neural Networks (GNNs) have emerged as a powerful approach for learning on graphs, as they enable the encoding of graph structure into neural network architecture to make predictions. While dating back to at least 2006, the first proposal for graph neural networks was published by Scarselli and Gori [8], and was later generalized in the 2008 paper "The Graph Neural Network Model" [9]. In the recent years, lots of progress has made GNNs a phenomenal and popular tool, gaining an increasing amount of interest with applications in Natural Language Processing (NLP) [10, 11], Computer Vision [12], Recommender System [13] and so on. From another viewpoint of graphs, GNNs have been successfully applied to analytical tasks such as link prediction [14] or node classification [15].

The capability of GNNs to acquire expressive graph representations is dependent on the quality and availability of graph-structured data. However, this poses a great challenge for most GNN architectures. On the one hand, if the underlying graph is already available, the performance of GNNs is vulnerable to any unnoticeable perturbation in graphs (adversarial attacks) [16, 17, 18]. The lack of robustness of these models can cause severe consequences related to safety and privacy. A considerable example is in credit card fraud detection, the fraudsters may carry out multiple transactions with a small group of high-credit users to conceal their activities, thus, easily bypass the GNNs-based detection system [19]. In addition, the assumption that the given graph is perfect does not necessarily hold true in reality because real-world graphs are oftenly noisy or incomplete. Another factor is the graph topology might merely represent physical connections, and fail to capture deeper implicitness among entities or cohorts of entities, which can be beneficial to infer information and optimize the downstream task. On the other hand, in numerous practical applications such as NLP or Computer Vision, it may be challenging to obtain the graph representation of the data. For instance, in the field of Computer Vision, the data is in the form of images, which require additional structure to depict relationships between objects present in the image. The manual graph construction approach requires huge human effort from domain experts to obtain a reasonable graph for further steps of building a model.

To tackle the above problems, the aim is to have a graph which is explicitly given, a setting referred to as latent graph. The Graph Structure Learning (GSL) methods aim to exploit latent graphs inferred from data for better performance to feed into GNNs. Another advantage of latent graphs is the potential to capture actual topology of structured data, thereby aiding in efficient processing and analysis. In addition, the latent graph obtained from the GSL allows the exploration and learning of the relationships between the behavioral influence among the nodes. Interestingly, sometimes the graph structure itself may be even more important than the downstream task due to the potential to convey interpretability of the model and consolidate the understanding of the data and model. For instance, analyzing the citation network, GSL techniques enable the identification of communities or clusters within the citation network. These communities represent groups of papers that are closely related and often share similar research topics. By discovering these communities, we gain insights into disciplinary boundaries, and interdisciplinary connections within the network.

With that motivation, the main aims of the thesis are:

- review and empirically validate the current Graph Structure Learning methods by focusing on their applicability in the citation network. Validation is conducted by assessing the methods' accuracy score on Cora dataset of Machine Learning publications, CiteSeer dataset with scientific publications, and PubMed dataset with scientific publications from PubMed database.

- analyze the mechanism of these approaches and analyze the influence of hyperparameters on model's behavior.

- discuss future work.

In this thesis, the sections are organized as follows. Section 2 provides background knowledge, which supports readers to easily move to the main focus of the thesis in upcomming sections. Section 3 starts by the preliminary review of graph neural networks, how graph structure learning techniques were studied and some recent methods, which are carefully examined on theoretical side in section 4 and empirical facet in section 5 with citation datasets. Section 6 highlights the related discussion on results and discussing . The thesis ends with section 7 covering the Github code where implementation can be found. The auxiliary parts are bibliography section, notations, figures and abbreviations notice used in the thesis.

2. Background

The background on basic graph theory, graph neural network will be covered in this section.

2.1. Graph Theory Basics

The aim of this section in the thesis is to equip readers with essential graph-related definitions that will enable them to comprehend subsequent sections, especially chapters 3 and 4.

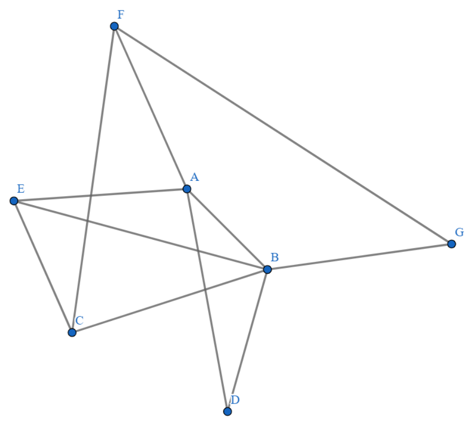

Figure 2.1: An illustration of an unweighted, undirected graph.

Definition 2.1 (Graph). A graph $\mathcal{G}$ is a triple $(\mathbf{V}, \mathbf{E}, \mathcal{A})$ where $\mathbf{V}$ is the set of vertices (aka nodes) with $\mathbf{V} = \{v_1, \ldots, v_N\}$, and $\mathbf{E} \in \mathbf{V} \times \mathbf{V}$ is the set of edges with $e_{i,j} = (v_i, v_j) \in \mathbf{E}$ to denote an edge pointing from v_j to v_i, and $\mathbf{A} = (a_{ij})$ is a (weighted) adjacency $n \times n$ matrix. The notation $\mathbf{A}$ is used to define the edges of the graph $\epsilon = \{(i, j) : a_{ij} > 0; i, j \in V\}$; the edge weight $a_{ij} \geqslant 0$ represents the affinity between v_i and v_j, where $a_{i,j} = 1$ if there is an edge from vertex v_i to vertex v_j and $a_{i,j} = 0$ otherwise. The neighborhood of a node v is defined as $N(v) = \{u \in \mathbf{V} | (v, u) \in \mathbf{E}\}$.

In the context of this thesis, given N data points of dimension d, a graph may have node attributes $\mathbf{X}$ (such graph is called attributed graph), where $\mathbf{X} \in \mathbb{R}^{N \times d}$ is a node feature matrix with $x_v \in \mathbb{R}^d$ representing a feature vector of a node v. Meanwhile, a graph may have edge attributes $\mathbf{X}^e$, where $\mathbf{X}^e \in \mathbb{R}^{m \times c}$ is an edge feature matrix with $x^e_{v,u} \in \mathbb{R}^c$ representing the feature vector of an edge (v, u).

With in the scope of graph application, often three terms related to graph type and its scale are considered. Popular types of graph are:

• **Directed/Undirected Graphs.** A directed graph is a graph with all edges directed from one node to another. Undirected graphs have edges, in which edges do not have a direction. A graph is undirected if an only if the adjacency matrix is symmetric. Also, edges in directed graphs provide more information than undirected graphs. Figure 2.2 shows undirected edge and directed edge.

Undirected edge
Directed edge

Figure 2.2: Undirected edge and directed edge [20].

• **Homogeneous/Heterogeneous Graphs.** A homogeneous graph, also known as a homogenous network, is a graph where all nodes are of the same type, and all edges are of the same type. A heterogeneous graph, also known as a heterogeneous network, is a graph where nodes and edges can have different types.

- For example, a transportation network consisting only of roads connecting locations would be examples of homogeneous graphs. But a citation network where nodes represent papers, authors, and conferences, and edges represent different types of relationships (such as citation, authorship, or conference attendance) would be an example of a heterogeneous graph.

• **Static/Dynamic Graphs.** A static graph is a graph that does not change over time. Once the graph is constructed, the set of nodes and edges remains fixed, and the graph does not evolve over time. And when input features or the structure of the graph vary with time, the graph is regarded as a dynamic graph.

- For example, static graphs include social networks, transportation networks, and citation networks that are constructed based on a snapshot of the network at a particular point in time. Examples of dynamic graphs include communication networks, financial transaction networks, and social networks where new nodes and edges can be added or removed over time.

Those categories can be combined, e.g. a research on static undirected heterogeneous graph. Related to graph scale, there is no clear threshold to define “small” or “large” graphs. The criterion is still changing due to the computability effectiveness, which changes over time.

2.2. Types of problems for Graph Structured Data

Given some graph-structured data in reality, there are often three predictions tasks needed to be performed on that graph: graph-level, node-level, and edge-level. For a graph-level task, a single property for a whole graph is the main target of prediction. In the case of a node-level task, some feature is predicted for each node in a graph. In an edge-level task, the property or presence of edges in a graph is predicted.

- **Graph-level tasks** have principal goal to predict a single property of an entire graph. For example, with classic MNIST image classification dataset, we want to label the number to an entire image; with NLP problem, we may want to conduct sentiment analysis for an entire sentence.
- **Node-level tasks** concern with tasks on node including node classification, node clustering, node regression, etc. Node classification is the task of categorizing identity of node into classes, and node regression predicts a continuous value for each node. For example, in the image segmentation, node-level prediction problems involve labeling each pixel in an image according to its role. Similarly, in text, predicting the parts-of-speech of each word in a sentence (such as noun, verb, adverb, etc.) is also a node-level task.
- **Edge-level tasks** deal with edge prediction. The tasks can be edge classification or predicting whether there is an edge connecting two nodes. For example, in a social network, link prediction can be used to suggest new friends or connections for a user based on their existing connections.

Some common useful problems can be formulated over graphs:

- Node classification: classifying individual nodes.
- Graph classification: classifying entire graphs.
- Link prediction: predicting the presence of connectivity between nodes.
- Node clustering: clustering together similar nodes based on the connecting edges.
- Influence maximization: determining influential nodes.

Another perspective can be taken into account is supervision, dealing with graph structured data, we can categorize tasks into three training settings, according to Zhou el at. [21]:

- **Supervised setting.** In supervised setting, labeled data is provided for training.

• **Semi-supervised setting** provides a limited number of labeled nodes while the majority of nodes are unlabeled for training. During test phase, the **transductive setting** requires the model to label the unlabeled nodes, while **inductive setting** provides new unlabeled nodes from the same distribution to infer. In the context of GNNs, **inductive setting** refers to the scenario in which the model is trained on a subset of the nodes and edges in a graph, but is evaluated on a different set of nodes and edges that were not seen during training; while **transductive setting** refers to the scenario in which the model is trained on a graph and is evaluated on a subset of the same graph that was not seen during training. **Transductive setting** is often easier than the **inductive setting** because the model has access to information about the entire graph during both training and evaluation.

• **Unsupervised setting** only provides unlabeled data for the model. Typical task of unsupervised setting is node clustering.

With different task type and training setting, specific loss function should be designed for the task. For instance, cross-entropy loss can be employed for the labeled nodes in the training set for a node-level semi-supervised classification task.

2.3. Graph Neural Networks

Graph neural networks are a novel class of neural network specifically designed to operate on graph-structured data, which can be used to solve the tasks we aforementioned, including graph-level, node-level, and edge-level tasks.

The mechanism of a message passing process proposed by Gilmer et al. [15] is shown by Battaglia et al. [22] about the ability of formulating most GNNs. The basic idea is that each node in the graph receives messages from its neighboring nodes, updates its own representation based on the received messages, and then passes on a new message to its neighbors. This process continues until each node has updated its representation based on information from its entire neighborhood. The final node representations can then be used for downstream tasks such as node classification or graph classification or edge-level tasks. The take-away idea of GNNs is "graph-in, graph-out" meaning GNNs accept a graph as input, with information about nodes, edges and graph topology, and progressively transform these embeddings into a new graph without changing the connectivity of the input graph.

The upcoming sections of the thesis will introduce two most important types of Graph Neural Network: Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT). Those two methods, one presumes every neighbor has the same importance, while other presumes some nodes are more essential than others.

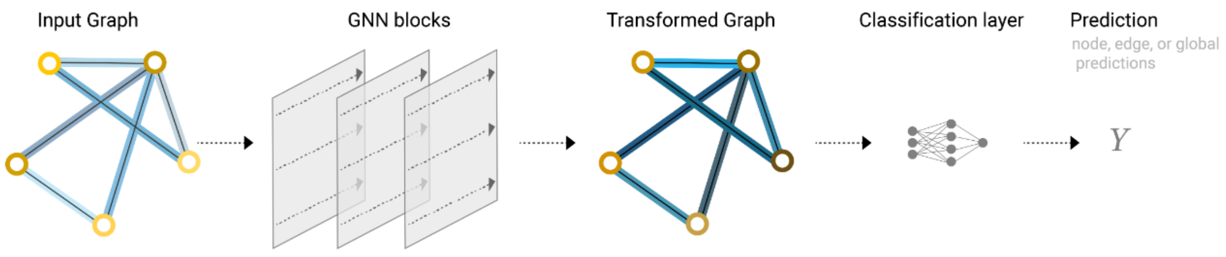

Figure 2.3: An end-to-end prediction task performed by GNN model [20]

2.3.1. Problem Formulation

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the goal is to learn a function which takes input:

- A feature description x_i for every node i; which is included in feature matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, where N is the number of nodes and d is the length of feature vector for each node.
- A matrix form representing for the description of the graph structure; typically an adjacency matrix $\mathbf{A}$.

and produces node-level output matrix $\mathbf{Z} \in \mathbb{R}^{N \times F}$, where F is the length of transformed featured vector of each node. The embedding $\mathbf{Z}$ is what GNNs attempt to find.

For a general $\mathrm{L}-\mathrm{th}$ neural network, we can mathematically formulate the structure $\mathbf{H}^{(l+1)}$ of $\mathrm{L}-\mathrm{th}$ layer as a result of a non-linear function:

$$\mathbf{H}^{(l+1)} = f_\theta(\mathbf{H}^{(l)}, \mathbf{A}), \tag{2.1}$$

where $\mathbf{H}^{(0)} = \mathbf{X}$ and $\mathbf{H}^{(L)} = \mathbf{Z}$.

2.3.2. Graph Convolutions

In a GCNs, a convolution operation involves aggregating information from a node's neighbors and then updating the node's feature representation. This process is typically done recursively over multiple layers to enable the model to capture more complex patterns and relationships between nodes.

The aggregation step in GCNs can be performed using various methods, such as mean or max pooling, attention-based mechanisms, or message passing algorithms [15]. Once the aggregation step is completed, a nonlinear function is applied to the aggregated information to obtain a new node representation, which is then passed to the next layer.

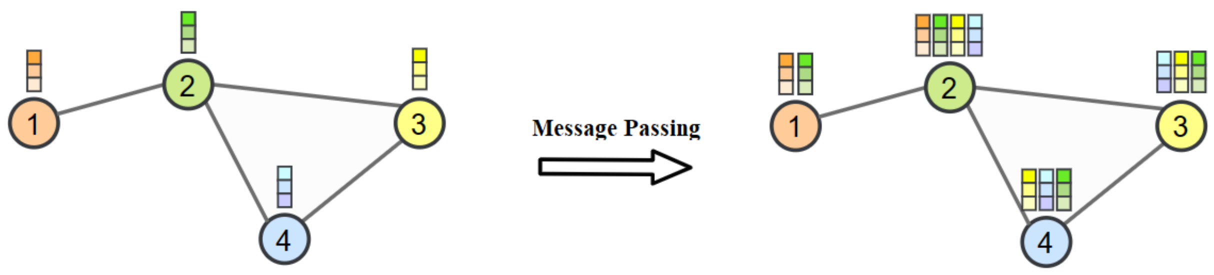

Figure 2.4: First two stages in message passing. Image from blog UvA DL Notebook.

In a typical message passing process, there are two stages. Stage 1, each node prepares a feature vector symbolizing the message it wants to send to its neighbors. Stage 2, the messages are sent to the adjacent neighbors. Stage 1 and 2 are visualized in illustration 2.4. Finally, we need to decide how to combine all the messages a node received. Since for different nodes, they have different number of adjacent neighborhoods, thus the number of message vary across nodes, we need an operation which works universally for any number of message. Usually, we often go to the sum or the mean. We will introduce the formula for the mean below. Given the previous features of nodes $\mathbf{H}^{(l)}$, the GCN layer is defined as follows:

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{2.2}$$

where $\mathbf{W}$ is the weight parameters with which we transform the input features into messages by calculating $\mathbf{H}^{(l)}\mathbf{W}^{(l)}$. From the adjacency matrix $\mathbf{A}$, we add the identity matrix in order to each node sends the it own message also to itself by the matrix $\hat{\mathbf{A}} = \mathbf{A}+\mathbf{I}$. Eventually, to take the average of messages instead of summing, we calculate the matrix $\hat{D}$ which is the diagonal matrix with D_{ii} is the number of neighbor nodes node i has. The function σ denotes any activation function, and it may not be the sigmoid function. Typically, GCNs use a ReLU-based activation function.

To extend the idea of feature exchanging beyond the scope between neighbor nodes, we can apply multiple GCN layers, which gives us the final layout of a GNN. This visualization 2.5 below (credit - Thomas Kipf, 2016) showcases how the GNN can be built up by a sequence of GCN layers and non-linear function as ReLU.

2.3.3. Latent Graphs

In certain scenarios, it may not be possible to obtain information about the underlying graph structure of a dataset. This is where the concept of learning the graph comes into play. The

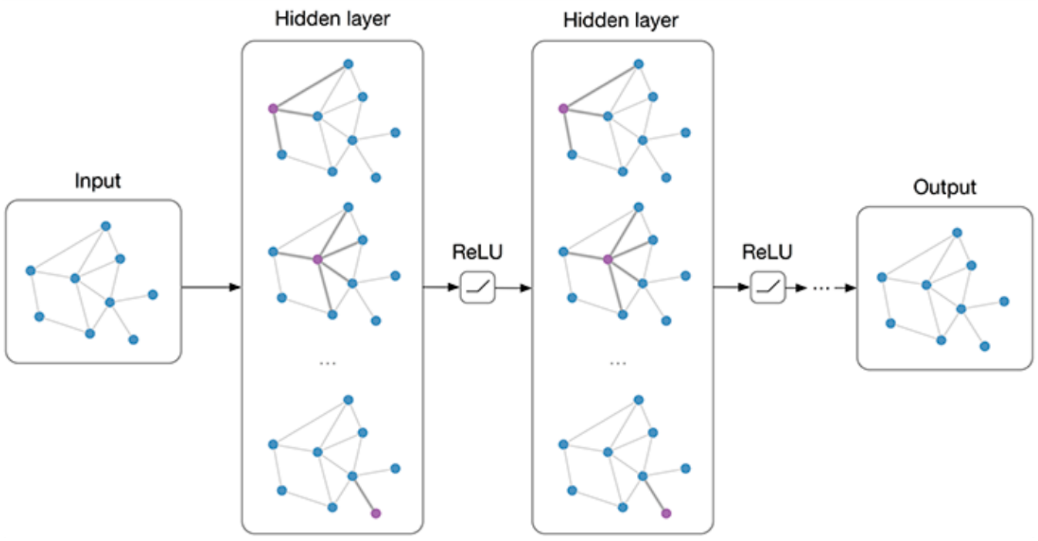

Figure 2.5: GNN built up by sequence of GCN layers and ReLU functions. Image from blog UvA DL Notebook

process of learning the graph involves developing a model that can represent the structure of the data, even when the true underlying graph is unknown. This serves two main purposes. Firstly, it provides a means of representing the structural relationships between data points, which can be used to guide the development of downstream applications. Secondly, the learned graph can be used as the basis for performing graph-based convolutions, which can be used to obtain embeddings or representations of the data points.

Overall, the ability to learn the graph in this way is a powerful, as it provides a way to work with complex datasets even when the underlying graph structure is unknown. This may lead to new insights to enhance the explainability of model's predictions and opportunities for applications in a wide range of fields.

3. Related work

In line with the focus of the thesis, we briefly introduce related work on Graph Neural Networks (GNNs), recent techniques in the field of Graph Structure Learning (GSL), in which we swiftly cover the GSL problem formulation, common GSL pipeline and different categories of GSL.

3.1. Graph Neural Networks

In the recent years, the successive achievements of deep neural networks have revolutionized many traditional machine learning tasks such as machine translation [23, 24], or object detection [25, 26]; which once heavily demanded manual feature engineering to extract usable feature sets, have transformed into a more end-to-end deep learning pipeline, by renowned techniques as convolutional neural networks (CNNs) [27], autoencoder [28], and recurrent neural networks (RNNs) [29]. Its success can be attributed to several key factors, including the availability of large-scale datasets, the rapidly developing computing hardware (e.g, GPU) and the effectiveness of new algorithms and frameworks of deep learning which extract latent representations from Euclidean data (e.g, images, text).

However, with the rise in applications where data is represented in the form of graphs, it has become evident that while deep learning is highly effective in capturing hidden patterns of Euclidean data, it may not be the optimal approach for graph data. To be more detailed, the assumption of independence amongst instances in the majority of existing deep learning algorithms does not necessarily hold true since neglection of links such as interactions, citations or friendships may cause significant problems. For instances, in e-commerce, the interactions between users and products are indispensable to make highly accurate recommendations, which can significantly better the user experience and drive sales. Another example is citation networks, where papers are linked to each other via citationships and need to be categorized into different groups. The complexity of those graph data has posed significant challenges for existing deep learning algorithms. Thus, graph neural networks (GNNs) have been developed as a branch of deep learning to effectively deal with the graph data.

Table 3.1: Taxonomy and representative publications of Graph Neural Networks. Table from [30]

Category		Publications
Reccurent Graph Neural Networks (RecGNNs)		[9, 31]
Convolutional GNNs (ConvGNNs)	Spectral methods	[32, 33, 34]
	Spatial methods	[35, 36, 37]
Graph Autoencoders (GAEs)	Network Embedding	[38, 39, 14]
	Graph Generation	[40, 41, 42]
Spatial-temporal GNNs (STGNNs)		[43, 44, 45]

Over the past few years, GNNs have inherited great success from neural networks to more effectively solve problems on graph data. One of the key strengths of GNNs is their ability to better both node features and graph topology to make predictions, which can significantly improve their accuracy compared to other machine learning algorithms.

Encouraged in the computer vision domain, it leads to a large number of methods re-defining the notion of convolution for graph data. Those approaches are in the branch of convolutional graph neural networks (ConvGNNs). See table 3.1. Popular mainstream of developing ConvGNNs are two families: spectral methods and spatial methods. The first family learns node representation based on graph spectral theory [32, 34]. The spectral methods involve analyzing the eigenvalues and eigenvectors of the graph Laplacian matrix to define filters that operate on the graph. In spectral methods, the convolution operation is defined in the spectral domain by using the Fourier basis of a given graph, which Bruna et al. [32] generalized. Spatial methods, on the other hand, directly operate on the graph structure by propagating information from neighboring nodes to update node features. Veličković et el. [17] introduced graph attention network (GAT) to learn different attention scores for neighbors when aggregating information. To gain a more thorough understanding of the topic, readers are encouraged to refer to the recent surveys available at [30, 21].

Despite numerous successful applications, most GNNs architectures still remain notable drawbacks which is the assumption that the underlying graph is given, fixed and unnoisy. With the message passing mechanism, a small noise will be propagated to neighborhoods and deteriorate the quality of many other connected nodes. Furthermore, as mentioned in section 1 about fraudulent detection in banking services, fake links to real accounts can be introduced which easily bypass the GNN-based detection system. In addition, some unnoticeable, deliberate perturbation (aka., adversarial attack) can heavily deteriorate performance of most GNNs [16]. Thus, there is an undeniable need for high-quality graph structure for GNNs to learn informative representations. On the application side, with limited or no prior knowledge, the predefined graph structures even constructed by domain experts can hardly carry complete information, in most cases, often hinder our understanding of the underlying mechanisms leading to the prediction. For example, the substructures (e.g, subgraphs or features) are vital to explain the prediction made by GNNs. For the healthcare domain, that model transparency is safety-critical to make predictions on patients. To work on explainable brain network analysis, beside predictions, the salient regions of interest contribute most to the prediction is also the main interest. The given graph structure often overlooks the underlying connection, which creates black boxes models without the ability to explain the predictions (anti-XAI). In a worse scenario, the graph itself may not be explicitly given, which gives no room to naïve applicability of GNNs.

The aforementioned issues motivate a considerable study on the topic of **Graph Structure Learning**, which manages scenarios where graph-structured data is noisy or unavailable. We will discuss in detail **Graph Structure Learning** in the upcoming section.

3.2. Graph Structure Learning for Graph Neural Networks

Graph Structure Learning (GSL) is a rapidly emerging research topic that has seen a significant number of interests in recent years [46, 47, 48, 49]. Applications of GSL are mushrooming in a wide range of domains. In the field of Bioinformatics, GSL is used to capture amino acids relation in a protein via the calculation between pairwise amino acids, as mentioned in the work of Guo et al. [50, 51]. For drug discovery, graph structures are a natural way to represent molecules, as atoms and bonds can be mapped into a molecular graph as vertices and edges as in the work of Fu et al. [52]. The main idea of GSL is to infer the underlying structure of a graph from observed data. The subsection of GSL is split into three parts: GSL problem formulation, common GSL pipeline for most existing GSL models and main categories of GSL.

3.2.1. GSL Problem Formulation

Let $\mathcal{G} = (\mathcal{A}, \mathcal{X})$ be a graph with N vertices, where $\mathcal{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix and $\mathcal{X} \in \mathbb{R}^{N\times F}$ is the node feature matrix with the i-th entry $x_i \in \mathbb{R}^F$ is the attribute of node v_i. When graph $\mathcal{G}$ is partially or totally unavailable, the target of GSL is to learn the adjacency matrix $\mathcal{A}^\star$ and its corresponding graph representation $\mathcal{Z}^\star \in \mathbb{R}^{N\times F'}$ that are optimized for certain downstream tasks.

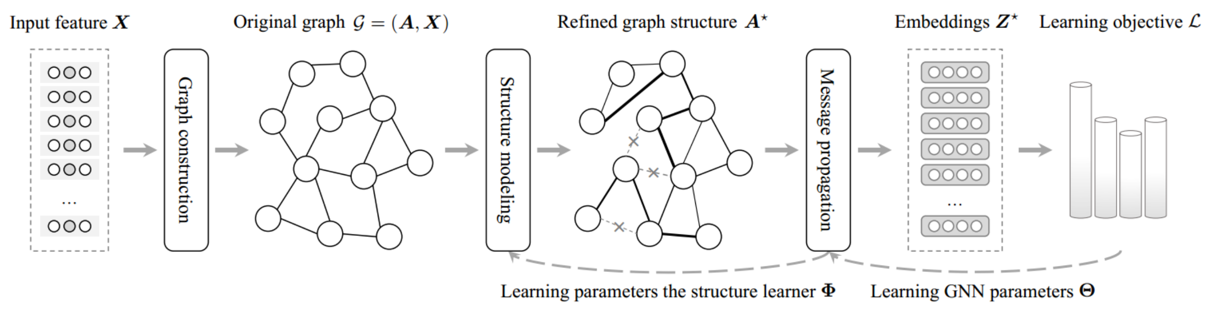
Input feature $\boldsymbol{X}$
Original graph $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X})$
Refined graph structure $\boldsymbol{A}^\star$
Embeddings $\boldsymbol{Z}^\star$
Learning objective $\mathcal{L}$
Graph construction
Structure modeling
Message propagation
Learning parameters the structure learner $\boldsymbol{\Phi}$
Learning GNN parameters $\boldsymbol{\Theta}$

Figure 3.1: A general pipeline of Graph Structure Learning [53].

Normally, GSL model have two trainable components: (1) a **GNN encoder** f_θ whose input is graph $\mathcal{G}$ and output is graph feature embeddings $\mathcal{Z}^\star$ used for downstream tasks and (2) a **structure learner** g_ϕ, computing the edge weights in the graph.

The training phase of model is to tuning $\{\theta, \phi\}$ with the learning objective [53]:

$$\mathcal{L} = \mathcal{L}_{tsk}(\mathcal{Z}^\star, Y) + \lambda \mathcal{L}_{reg}(\mathcal{A}^\star, \mathcal{Z}^\star, \mathcal{G}). \tag{3.1}$$

where the first term $\mathcal{L}_{tsk}$ relates to a task-specific objective which Y is the ground truth, the second term $\mathcal{L}_{reg}$ learns the optimal graph structure $\mathcal{A}^\star$ and its feature representations $\mathcal{Z}^\star$, and λ is a hyperparameter tuned by the model trainer.

3.2.2. Common GSL Pipeline

A general pipeline of Graph Structure Learning in Figure 3.1 proposed by Zhu, Yanqiao, et al [53] follows three main stages: (1) graph construction, (2) graph structure modelling, and (3) message propagation.

Stage 1: Graph Construction

In many real-world dataset, in which the graph structure is incomplete or even not given at all, we need to build preliminary graph as a starting point. The most two common ways to build the preliminary graph are (1) k Nearest Neighbots (kNN graphs) [54] and (2) ε proximity (ε-graphs) [55]. The main idea of those two methods is to compute the pairwise distance of node feature using a defined function. For the kNN graphs, two nodes v_i and v_j are connected if v_i is in the top k-closest neighbors of v_j. For the ϵ-graphs method, an edge is created between two nodes if a predefined metric of distance is smaller than preset threshold ε.

Stage 2: Graph Structure Modeling

With the constructed preliminary graph, the task of stage 2 is to use the structure learner g to update the edge connectivity. There are three groups:

1. **Metric-based approaches:** we embed node features them employ a metric function to calculate edge weights on pairwise node embeddings.

2. **Neural approaches**: those methods utilize the power of more complex neural networks to assign more appropriate edge weights based on the features of the nodes that are connected by those edges.

3. **Direct approaches:** from the preliminary graph, we obtain the adjacency matrix. In those approaches, we treat that matrix as learnable parameters and simultaneously optimize them with GNN parameters θ.

The first two approaches, neural and metric-based approaches, performed an embedding/ neural network to learn edge weights. The embedding or neural network can be treated as parameterized networks, which produces a matrix $\tilde{\mathcal{A}}$ representing the optimized graph structure. With the matrix $\tilde{\mathcal{A}}$, extra postprocessing steps may be involved to obtain the final graph $\mathcal{A}^\star$ (e.g: by using discrete sampling [56]).

Stage 3: Message Propagation

Once an optimized adjacency matrix $\mathcal{A}^\star$ is obtained, node features are propagated to the refined neighborhoods using a GNN encoder f, resulting in embedding $\mathcal{Z}^\star$.

Remarkably, stage 2 and stage 3 are repeated until specified criteria are met as optimizing the graph structure is not an easy task. To clarify, the representation obtained from stage 3 will be optimized to model edge connectivities in stage 2, in an iterative manner.

3.2.3. Current GSL Approaches

From different perspectives, multiple Graph Structure Learning (GSL) approaches have been developed. This section aims to present prominent categories for the state-of-the-art techniques in GSL. Each category will be demonstrated by a GSL method which is further discussed in detail in section 4.

2.2.3.1 Attention-based Graph Learning

Some related work included in this category deal with attention to the nodes and edges, since edge-based attention can be seen as a latent structure learning process. In attention-based graph learning, the model learns to focus on certain parts of the graph while ignoring others. This is done using **graph attention networks (GAT)** [17], which allows the model to assign weights on the edges of a fixed graph to learn the importance of each neighbor and also be less likely prone to noisy neighbor nodes . These weights are used to compute a weighted sum of the features of the neighboring nodes and edges, which is then used to update the features of the current node. Newly introduced method can be named such as Gated Attention Network (GAAN) [57], RNN-based attention models [58, 47]. Our primary area of investigation is GAT framework, which will be further discussed in section 4.

However, a typical limitation of these attention-based graph learning methods is their dependence on an input graph. Additionally, attention is calculated directly on edge features, necessitating sparsity in the graph to guarantee computational feasibility, especially for bigger graphs.

2.2.3.3 Learning Discrete Graph Structures

The key idea of these learning discrete graph structures techniques is the consideration the graph structure as a random variable, from which a discrete graph structure can be obtained by sampling a probabilistic adjacency matrix [59]. These techniques usually leverage various techniques such as bilevel optimization [60] or reinforcement [61] to jointly optimize the graph structure and GNN parameters. It is worth mentioning that these models are frequently restricted to the transductive learning settings, wherein the node features and graph structure are fully observed throughout both the training and inference stages.

The most noticing method in this category with reinforcement is Kazi's work of [61]. With the provided related work of Graph Convolutional Networks and Graph Structure learning, we have completed the Related work chapter, it is now time to proceed to chapter of Methodology of the thesis.

4. Methodology

In this section, as mentioned in the main aim of the thesis, we will review some current Graph Structure Learning (GSL) methods. Those recent noteworthy GSL techniques include: Graph Attention Networks - GAT and Differentiable Graph Module (DGM) architecture.

As stated in the primary objective of this thesis, this section will focus on examining some of the most recent and graph structure learning (GSL) methods. The review will cover cutting-edge GSL techniques, including Graph Attention Networks (GAT), Differentiable Graph Module (DGM) for Graph Neural Networks. In particular, detailed reviews on those methods will be provided consecutively. By doing so, readers will be provided with solid comprehension about those recent GSL methods.

4.1. Graph Attention (GAT)

Differ from the idea of equally treating messages received from the neighbor nodes, attention depicts a weighted average of multiple elements with the weights computed based on input query and elements's keys. In the context of graph, the method of attention is called Graph Attention Network (GAT), proposed by Velickovic et al. [17]. The first few steps of GAT are similar to GCNs, in which the *graph attention layer* creates a message for each node by using the weight matrix. For the attention part, it uses the message from itself's node as a query, and the messages to average as both keys and values including the message to itself. Mathematically, we can formulate the problem as below:

A single *graph attention layer* takes input is a set of node features, $\mathbf{h} = \{\overrightarrow{h_1}, \overrightarrow{h_2}, ..., \overrightarrow{h_N}\}, \overrightarrow{h_i} \in \mathbb{R}^F$, where N is the number of nodes, and F is the length of feature vector in each node. The *graph attention layer* produces a new set of node features (of potentially different $F' \neq F$), $\mathbf{h}' = \{\overrightarrow{h'_1}, \overrightarrow{h'_2}, ..., \overrightarrow{h'_N}\}, \overrightarrow{h'_i} \in \mathbb{R}^{F'}$.

As a way of transforming the input features into higher-level features, we need a learnable linear transformation. Initially, a *weight matrix* $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is applied to each node. We then perform *self-attention* on the nodes $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \longrightarrow \mathbb{R}$ to obtain the attention coefficients

$$e_{ij} = a(\mathbf{W}\overrightarrow{h_i}, \mathbf{W}\overrightarrow{h_j}) \tag{4.1}$$

indicating the importance of node j's features to node i's features. Noticing that we usually use a the weight matrix of the MLP for a, which has the shape of $[1, 2 \times F']$. Then the formulation becomes:

$$e_{ij} = a(\mathbf{W}\overrightarrow{h_i} \| \mathbf{W}\overrightarrow{h_j}), \tag{4.2}$$

the operation $\|$ is the column-wise concatenation.

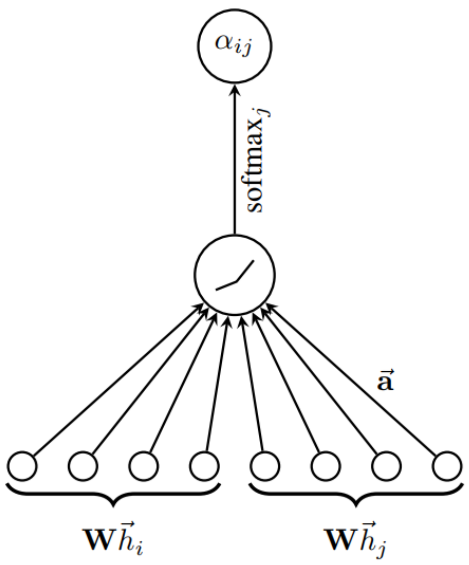

Figure 4.1: The attention mechanism $a(\mathbf{W}\overrightarrow{h_i}, \mathbf{W}\overrightarrow{h_j})$, with a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation [17].

The most general formulation of GAT (which is rarely used) is the model allows every node to send message to all other nodes, which drop the information of structure. In the paper of Velickovic et al. [17], the technique is restricted to one-hop neighbor, which is called *masked attention* - where we only compute e_{ij} for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the set of neighborhood of node i in the graph. Later on, when we mention, we mention only about one-hop neighbors of i (including i). For the purpose of comparing across different nodes, we normalize them across all choices of j using the softmax function:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \tag{4.3}$$

In the thesis, we employ the attention mechanism a by a weight vector $\overrightarrow{a} \in \mathbb{R}^{2F'}$, then, apply the LeakyReLU (Rectified Linear Unit) with input slope $\alpha = 0.2$; where the general form of LeakyReLU is:

$$\text{LeakyReLU}(\mathbf{x}) = \max(\alpha * \mathbf{x}, \mathbf{x}), \tag{4.4}$$

where α is a small constant. With $\alpha = 0$, we have the original ReLU function. We fully expand the equation 4.3, the coefficients computed may then be expressed as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T[\mathbf{W}\overrightarrow{h_i}\|\mathbf{W}\overrightarrow{h_j}]))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\vec{a}^T[\mathbf{W}\overrightarrow{h_i}\|\mathbf{W}\overrightarrow{h_k}]))}. \tag{4.5}$$

where $.^T$ represents transposition. What we have done in equation 4.5 can be visualized in figure 4.1.

$$\alpha_{ij} = \frac{exp(\vec{a}^T[\mathbf{W}\overrightarrow{h_i}\|\mathbf{W}\overrightarrow{h_j}])}{\sum_{k\in\mathcal{N}_i} exp(\vec{a}^T[\mathbf{W}\overrightarrow{h_i}\|\mathbf{W}\overrightarrow{h_k}])} = \frac{exp(\vec{a}^T_{:,:F'}\mathbf{W}\overrightarrow{h_i} + \vec{a}^T_{:,F':}\mathbf{W}\overrightarrow{h_j}])}{\sum_{k\in\mathcal{N}_i} exp(\vec{a}^T_{:,:F'}\mathbf{W}\overrightarrow{h_i} + \vec{a}^T_{:,F':}\mathbf{W}\overrightarrow{h_k})} \tag{4.6}$$

while

$$\frac{exp(\vec{a}^T_{:,:F'}\mathbf{W}\overrightarrow{h_i} + \vec{a}^T_{:,F':}\mathbf{W}\overrightarrow{h_j}])}{\sum_{k\in\mathcal{N}_i} exp(\vec{a}^T_{:,:F'}\mathbf{W}\overrightarrow{h_i} + \vec{a}^T_{:,F':}\mathbf{W}\overrightarrow{h_k})} = \frac{exp(\vec{a}^T_{:,F':}\mathbf{W}\overrightarrow{h_j})}{\sum_{k\in\mathcal{N}_i} exp(\vec{a}^T_{:,F':}\mathbf{W}\overrightarrow{h_k})} \tag{4.7}$$

The reason why we apply the non-linearity before softmax over elements is because if we don't have the non-linearity, the attention term with h_i cancels out itself, resulting in the attention being independent of the node itself as in equation (4.6) and (4.7).

Once we obtain all attention indicators α_{ij}, we calculate the output feature for each node by calculating the weighted average:

$$\mathbf{h_i'} = \sigma(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}h_j), \tag{4.8}$$

where σ is another non-linearity. The full version of graph attention network is to multiple-head attention resulting in N attention layers in parallel. Visually, we can illustrate the whole process of multi-head graph attentional layer in this figure 4.2 (Figure credit [17]). More detail of multi-head attentional layer can be found at [17].

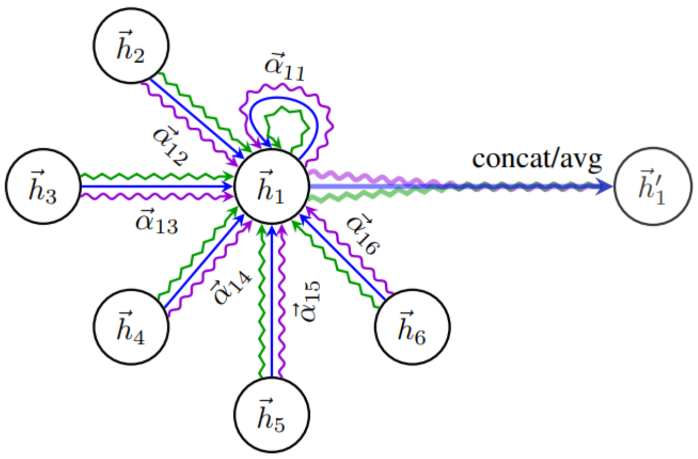

Figure 4.2: Multi-head attention with K = 3 heads by node 1 on its neighborhood. Three different colors of arrows (green, blue, and purple) that are afterward concatenated. Figure credit [17].

4.2. Differentiable Graph Module (DGM) architecture

The DGM architecture [61] relies on optimizing the network parameters and utilizing the output features of each layer. The DGM architecture is comprised of two key blocks, the **Differentiable Graph Module (DGM)** and the **Diffusion Module**. In the scope of the thesis, only discrete DGM (dDGM) structure will be examined. The algorithm 1 and figure 4.3 can help comprehend the primary concept of DGM architecture (Figure credit [61]).

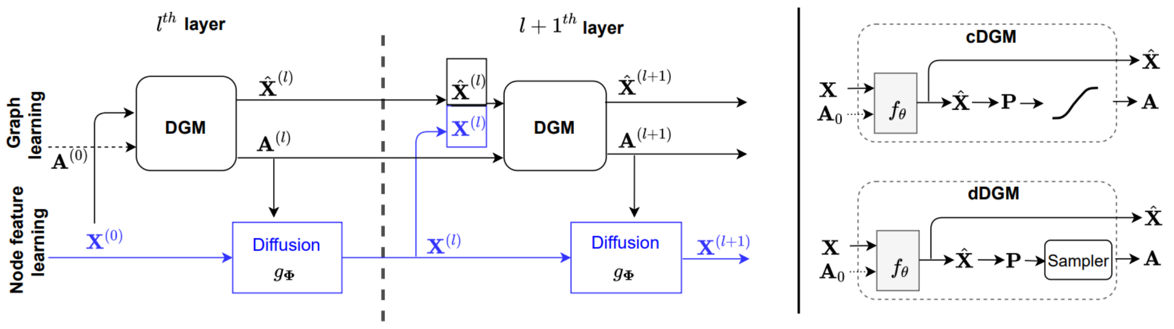

Figure 4.3: Left: The architecture is comprised of two layers: the Differentiable Graph Module (DGM) responsible for learning the graph, and the Diffusion Module which utilizes graph convolutional filters. Right: Two variants of DGM, cDGM and dDGM.

Algorithm 1: Algorithm for full model that produces predicted labels $\hat{\mathbf{Y}}$ which then used by the loss function to train the model. (a_{ij}) denotes the matrix $\mathbf{A}$, while $\mathbf{p_i}$ refers to the $i-th$ row of P. f_Θ and g_Φ are generic function with the learnable parameters Θ and Φ respectively. MLP is the Multi Layer Perceptron used for classification with parameters Ψ.

```
Data: $\mathbf{X}, \mathbf{A}$
Output: $\hat{\mathbf{Y}}$
1  begin
2      $\mathbf{X}^{(0)} \longleftarrow \mathbf{X}$
3      $\mathbf{A}^{(0)} \longleftarrow \mathbf{A}$
4      $\hat{\mathbf{X}}^{(0)} \longleftarrow \emptyset$
5      for $i \in 0...L-1$ do
6          $\hat{\mathbf{X}}^{(l+1)}, \mathbf{A}^{(l+1)} \longleftarrow \text{DGM}([\mathbf{X}^{(l)}|\hat{\mathbf{X}}^{(l)}], \mathbf{A}^{(l)}, f_\Theta^{(l)}, t^{(l)}, T^{(l)})$
7          $\mathbf{X}^{(l+1)} \longleftarrow g_\Phi^{(l)}(\mathbf{A}^{(l+1)}, \mathbf{X}^{(l)})$
8      $\hat{\mathbf{Y}} \longleftarrow \text{MLP}_\Psi(\mathbf{X}^{(L)})$
9  Function DGM($\mathbf{X}, \mathbf{A}, f_\Theta, t, T$):
10     $\hat{\mathbf{X}} \longleftarrow f_\Theta(\mathbf{X})$
11     $p_{ij} \longleftarrow e^{-t\Delta(\hat{\mathbf{X}}_\mathbf{i}, \hat{\mathbf{X}}_\mathbf{j})^2}$
12     switch sampling do
13         case discrete do
14             for $i \in 1...N$ do
15                 $\mathbf{q} \sim U(0,1)$
16                 $\mathbf{j}_{\{k\}} = \text{argtopK}(\log(\mathbf{p}_i) - \log(-\log(\mathbf{q})))$
17                 $a_{ij} = \begin{cases} 1, & \text{if } j \in \mathbf{j}_{\{k\}} \\ 0, & \text{otherwise} \end{cases}$
18     return $\hat{\mathbf{X}}, (a_{ij})$
```

4.2.1. Differentiable Graph Module (DGM)

- **Target**: build the (weighted) graph representing the input space.
- **Input**: node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$ [2.1] (in case domain knowledge is absent) or initial graph $\mathcal{G}_0$ with initial edge weights between nodes defined by domain experts.

- **Output**: graph $\mathcal{G}$.

Notice that, since graph $\mathcal{G}$ and graph $\mathcal{G}_0$ are fixed, the two graphs then can be represented by their adjacency matrices, $\mathbf{A}$ and $\mathbf{A}_0$. The inner function of DGM starts by transforming the input features $\mathbf{X} \in \mathbb{R}^{N \times d}$ into auxiliary features $\hat{\mathbf{X}} = f_\Theta(\mathbf{X}) \in \mathbb{R}^{N \times \hat{d}}$ by means of a parametric function f_Θ, in order to reduce the input dimension ($\hat{d} \ll d$). In case $\mathcal{G}_0$ is provided, new features $\hat{\mathbf{X}}$ are calculated by edge-or graph convolution on $\mathcal{G}_0$. From the auxiliary features $\hat{\mathbf{X}}$ constructed, we define the edge probabilities as:

$$p_{ij}(\mathbf{X}, \Theta, t) = e^{-t\Delta(\hat{\mathbf{x}_i}, \hat{\mathbf{x}_j})^2} = e^{-t\Delta(f_\Theta(\mathbf{x}_i),\ f_\Theta(\mathbf{x}_j))^2},$$

where t is a learnable parameter and $\Delta(\cdot,\cdot)$ is distance between two nodes in the graph embedding space. We will investigate two metrics, Euclidean and Hyperbolic metrics.

The following provides detailed information about the discrete sampling (dDGM) strategy used in the DGM module.

4.2.1.2 Discrete sampling (dDGM):

This discrete sampling strategy wil use Gumbel-Top-k-trick [56] to sample edges from the probability matrix $\mathbf{P}(\mathbf{X}; \Theta, t)$. For each node i, we extract k edges $(i, j_{i,1}), ..., (i, j_{i,k})$ as the first k elements of

$$\text{argsort}(\log(\mathbf{p}_i) - \log(-\log(\mathbf{q})),$$

where $\mathbf{q} \in \mathbb{R}^N$ is uniform i.i.d in the interval $[0, 1]$. The edge set of the sparse graph $\mathcal{G}$ is constructed as

$$\mathcal{E}(\mathbf{X}; \mathbf{\Theta}, \mathbf{t}) = \{(i, j_{i,1}), ..., (i, j_{i,k}) : i = 1, ...N\}.$$

The key advantage of dDGM is the adjacency matrix $\mathbf{A}$ of graph $\mathcal{G}$ is sparse, which means lower computational and memory needed for the diffusion operation.

4.2.2. Diffusion Module

The diffusion module takes the adjacency matrix $\mathbf{A}$ produced by DGM and the feature $\mathbf{X}^{(l)}$ as inputs, and produce a new set of features $\mathbf{X}^{(l+1)} = g_\Phi(\mathbf{X}^{(l)})$ as shown in figure 4.3. We can consider g_Φ as either edge- or graph-convolution on $\mathbf{A}$.

4.2.3. DGM classification model

In the DGM classification model, we construct a multi-layer network, with L layers, numbered as $l = 1, ..., L$. Each layer comprises a **DGM** and a **Diffusion Model**, as shown in Figure 4.2.

The $l-th$ layer of the architecture works as below:

$$\hat{\mathbf{X}}^{(l+1)} = f_{\Theta}([\mathbf{X}^{(l)}|\hat{\mathbf{X}}^{(l)}], \mathbf{A}^{(l)}) \tag{4.9}$$

$$\mathbf{A} \sim \mathbf{P}(\hat{\mathbf{X}}^{(l+1)}) \tag{4.10}$$

$$\mathbf{X} = g_{\Phi}(\mathbf{A}^{(l+1)}, \mathbf{X}^{(l)}) \tag{4.11}$$

We then assume $\mathbf{X}^{(0)} = \mathbf{X}$. If the knowledge of the structure of the data is unavailable, then $\mathbf{A}^{(0)} = \mathbf{I}$ (i.e, the initial graph is $\mathcal{G}^{(0)} = (\mathbf{V}, \emptyset)$) and $f_{\Theta}^{(0)}$ is a node-wise function (MLP). In case of classification task, the final node features $\mathbf{X}^{(L)}$ of last layer L can then be given as input to a MLP to obtain the final node predictions. This end-to-end style model can be defined as $\widehat{Y} = \mathbb{M}(X^0, A^0)$ where, $\widehat{Y}$ is predicted label vector.

Loss function and Training

In the dDGM framework [61], a compound loss rewards edges involved in a correct classification and penalizes edges leading to misclassification was introduced. Let $\mathbf{y}$ denote the vector of node-wise labels predicted by the model at step (t) and $\tilde{y}$ the groundtruth labels. The reward function is defined:

$$\delta(y_i, \tilde{y_i}) = \mathbb{E}((a_i)) - a_i \tag{4.12}$$

as the difference between the average accuracy of the $i-th$ sample and the current success value $a_i = 1$ if $y_i = \tilde{y_i}$ and 0 otherwise. The graph loss is derived as:

$$L_{graph}(\Theta^{(1)}, ..., \Theta^{(L)}) = \sum_{\substack{i=1..N \\ l=1...L \\ j:(i,j)\in\mathcal{E}^{(l)}}} \delta(y_i, \hat{y_i}) \log p_{ij}^{(l)}(\Theta^{(l)}) \tag{4.13}$$

whose gradient approximates the gradient of the expectation:

$$\mathbb{E}_{(\mathcal{G}^{(1)},...,\mathcal{G}^{(L)})\sim(\mathbf{P}(\Theta^{(1)}),...,\mathbf{P}(\Theta^{(L)}))}\sigma_i\delta(y_i, \tilde{y_i}) \tag{4.14}$$

with respect to the parameters of the graphs in all the layers.

Then, the estimation $\mathbb{E}(\mathrm{a}_i)$ is calculated with two different strategies depending on the application. When the node set remains fixed for each sample and during training, exponential moving average strategy on accuracy values during training process will be adopted. In this case, $\mathbb{E}(a_i)^{(t+1)} = \alpha\mathbb{E}(a_i)^{(t)} + (1-\alpha)a_i$, with $\alpha = 0.95$ in all experiments. If node set changes during traning or between different samples, $\mathbb{E}(a_i)$ will be estimated individually for each training step (t) evaluating the model multiple times on different sampled graphs.

5. Experiments and results

This section presents experiments conducted on three standard citation network benchmark datasets: Cora, CiteSeer, and PubMed, aimed at predicting category for each publication.

5.1. Datasets

In this section, information will be provided about the datasets used in the experiments aimed at comparing different frameworks for graph structure learning across various settings, thereby enabling a comprehensive evaluation of their efficacy and performance.

1. Cora. The Cora dataset consists of 2708 Machine Learning publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

2. CiteSeer. The CiteSeer dataset consists of 3312 scientific publications classified into one of six classes. The citation network consists of 4732 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3703 unique words.

3. PubMed. The Pubmed dataset consists of 19717 scientific publications from PubMed database pertaining to diabetes classified into one of three classes. The citation network consists of 44338 links. Each publication in the dataset is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words.

Each of these datasets consists of a single graph in which nodes correspond to documents and undirected edges to citation links. Each node has a class label. The goal is to predict the category of each document in **transductive setting**. Honoring the transductive setup, the training algorithm has access to all the nodes' feature vectors, and the category of some nodes is not known during training. Additional information regarding these datasets is provided in Table 5.1. For the distribution of node degrees in each dataset, reference to Figure 5.1.

Table 5.1: Summary of the datasets used in experiments.

	Cora	**Citeseer**	**Pubmed**
Task	Transductive	Transductive	Transductive
# Nodes	2708 (1 graph)	3327 (1 graph)	19717 (1 graph)
# Edges	5429	4732	44338
# Features/Node	1433	3703	500
# Classes	7	6	3
# Training Nodes	140	120	60
# Validation Nodes	500	500	500
# Test Nodes	1000	1000	1000
Isolated nodes	No	Yes	No
Having loops	No	No	No
Edges are directed	No	No	No

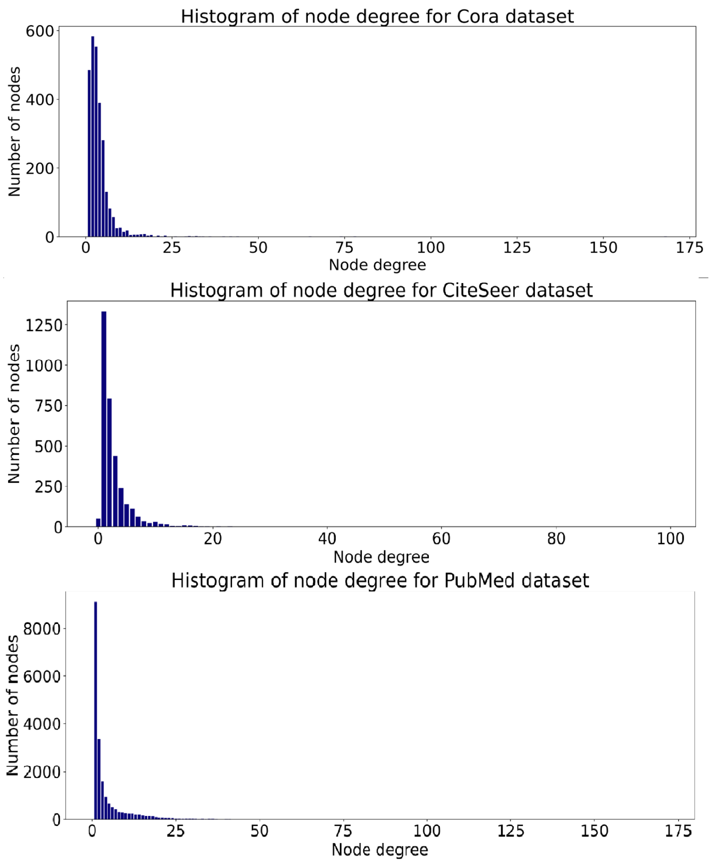

Figure 5.1: Histograms of node degree for Cora, CiteSeer and PubMed datasets

5.2. Experimental setup

All of the experiments for comparisons will be conducted in transductive settings. In the first stage, four frameworks including MLP (multilayer perceptron), Graph Convolution Network (GCN), Graph Attention Network (GAT) and discrete-DGM (d-DGM) are tested.

5.2.1. Architecture details

1. MLP architecture: two linear layers with ReLU activation, dropout regularization and Adam optimizer with learning rate of 0.01. The mean and standard deviation are reported in accuracy (in percent) for 10 times of running.

2. GCN architecture: two graph convolutional layers with ReLU activation, dropout regularization and Adam optimizer with learning rate of 0.01. The mean and standard deviation are reported in accuracy (in percent) for 10 times of running.

3. GAT architecture: two graph attention layers with ELU activation, dropout regularization and Adam optimizer with learning rate of 0.01. The mean and standard deviation are reported in accuracy (in percent) for 10 times of running.

4. d-DGM architecture: one DGM layer where graph is given as input to the associated diffusion layer. The output is passed through a final MLP of size (8, 8, c), which classifies each node into one of c classes. DGM layer function f is composed by two GCN layers with output feature dimensions of 16 and d respectively. $d = 4$ was set as the default graph embedding space dimension. The diffusion function g is composed of three GCN layers with output sizes 32, 16 and 8, respectively. The sample kNN graph was set of $k = 5$ neighbors. Before being processed by the DGM layer, input features are transformed by a perceptron (a linear transformation followed by a ReLU activation). Adam optimizer was used with learning rate of 0.01, and report the mean and standard deviation in accuracy (in percent) for 10 times of running.

5.3. Results

5.3.1. Comparisons for frameworks

The results of comparative evaluation for four frameworks are summarized in table 5.2. Overall, we can see a significant improvement in accuracy between the MLP model and graph-based

Table 5.2: Classification accuracies for Cora, Citeseer and PubMed in transductive setting.

Method	**Cora**	**Citeseer**	**Pubmed**
MLP	52.83 ± 0.96%	50.59 ± 1.35%	71.11 ± 1.44%
GCN	80.35 ± 0.68%	66.85 ± 0.84%	78.56 ± 0.55%
GAT	78.98 ± 1.03%	68.89 ± 0.81%	75.51 ± 1.70%
d-DGM	**81.52 ± 1.76%**	**70.47 ± 1.12%**	**86.98 ± 1.16%**

models. For example, with Cora and Citeseer and Pubmed dataset, the mean accuracy was improved averagely 30%, 20% and 10%. In detail:

1. Multilayer Perceptron (MLP): exhibits the lowest accuracy scores among the four frameworks on all three datasets. In comparison to the other methods, it performs relatively poorly, particularly on the Citeseer and Cora datasets.

2. Graph Convolutional Network (GCN): demonstrates a significant improvement over MLP on all datasets, resulting in higher accuracy scores. It consistently outperforms MLP across all three datasets, highlighting the effectiveness of graph convolutional layers.

3. Graph Attention Network (GAT): slightly outperforms GCN on the Cora dataset but falls behind on the Citeseer and Pubmed datasets. Although it shows comparable results to GCN, it does not consistently outperform it.

4. discrete-DGM (dDGM): achieves the highest accuracy scores among the four frameworks on all three datasets. It surpasses both GCN and GAT, indicating the effectiveness of the discrete-DGM approach for these tasks.

The main cause for the difference between performances can be explained by the architectural structure, with the 3 graph-based datasets, the MLP architecture takes no notice of the links between nodes, only the information of node features were taken. With GCN, GAT and d-DGM, the methods of aggregating information are different, which can be explained swiftly:

- GCN: in the process of aggregating information and update the feature vector of the current node, GCN assumes every node has **the same importance**.

- GAT: the assumption of the same importance does not necessarily hold true, since some nodes are more essential than others. In the process of aggregating information and update the feature vector of the current node, GAT assumes **some nodes are more important than the other nodes**, then assigns a weighting factor to every connection.

- d-DGM: the process of aggregating information (with and without assigning weighting factor) does not necessarily perform better, since in some cases, some included noisy nodes may send information to the current nodes for updating current feature vector of the current node. In the process of aggregating information and update the feature vector of the current node, d-DGM takes only k most influential neighboring nodes to join in the process of updating feature vector of the current node. This style is able to reduce noisy nodes affecting the current node.

Overall, according to the results presented in table 5.2, the results suggest that discrete-DGM is the most effective framework among the four, consistently achieving the highest accuracy scores. GCN performs well and outperforms MLP, while GAT performs similarly to GCN but may be slightly less effective on certain datasets. Therefore, if accuracy is the primary concern, discrete-DGM is the recommended choice, followed by GCN and GAT, while MLP (traditional neural network) should be avoided for these particular tasks of dealing with graph-based datasets. In the upcoming experiments, various hyperparameter tuning options will be explored to further maximize the potential and effectiveness of the d-DGM framework.

5.3.2. Framework d-DGM exploration and exploitation

In this section, different settings of hyperparameters of d-DGM framework will be examined.

5.3.2.1 Graph sparsity - kNN

In this experiment, examination the sparsity of the graph is conducted. Sparsity of the graph is influenced by the choice of the hyperparameter k when constructing the kNN graph. The purpose of this experiment is to assess the performance of the graph learning module at different sparsity levels and determine a suitable range of sparsity in terms of the hyperparameter k. As shown in Table 5.3, the value of this hyperparameter has a significant impact on the network's performance. Across all the datasets considered, we find that a favorable range for k is between 3 and 5. This suggests that the relationships within the latent network can be effectively captured by a highly sparse graph. Consequently, the discrete sampling strategy for DGM achieves high efficiency in this scenario.

5.3.2.2 Graph embedding space geometry

In this experiment, the objective is to examine whether embedding graph nodes in a hyperbolic space yields benefits for node classification rather than standard Euclidean. The distance in hyperbolic space is defined as:

Table 5.3: Performance of dDGM under different choices of k-nn when sampling the graph. The accuracy is reported with mean and standard deviation on 10 runs.

k	Cora	Citeseer	Pubmed
1	52.04 ± 2.77%	54.14 ± 3.42%	**87.84 ± 0.72%**
3	**81.62 ± 2.23%**	71.3 ± 1.41%	**87.84 ± 1.58%**
5	75.5 ± 15.14%	**71.42 ± 2.10%**	87.72 ± 0.67%
10	77.16 ± 2.72%	68.86 ± 1.37%	87.16 ± 1.20%
20	67.68 ± 12.79%	67.84 ± 2.47%	86.80 ± 1.34%

$$\mathrm{d}(\mathbf{x},\mathbf{y}) = \cosh^{-1}\left(1 + 2\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{(1-\|\mathbf{x}\|_2^2)(1-\|\mathbf{y}\|_2^2)}\right), \tag{5.1}$$

where two points are inside the unitary open ball, $\mathbf{x}, \mathbf{y} \in \mathcal{B}^d$. The norm $\|.\|_2^2$ is the standard euclidean squared norm.

In case of the Euclidean latent space geometry, the distance is defined as $\mathrm{d}(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2^2$.

Table 5.4: Performance reported with euclidean and hyperbolic norms. The mean and standard deviation of accuracy on 10 runs are reported.

	Cora	Citeseer	Pubmed
Euclidean	**81.44 ± 1.32%**	**71.42 ± 2.1%**	**87.72 ± 0.67%**
Hyperbolic	80.18 ± 2.13%	70.1 ± 2.0%	81.82 ± 14.31%

Table 5.4 presents the performance of models that learned a graph assuming either a hyperbolic or Euclidean latent space. Regarding dDGM, the results reveal that the model with a Euclidean embedding consistently outperforms the hyperbolic embedding across the three investigated datasets. Additionally, the standard deviation indicates that the Euclidean space offers greater stability compared to the hyperbolic space. These findings suggest that the considered datasets do not derive any advantages with the hyperbolic latent space.

5.3.2.3 Graph embedding space dimension

A larger dimension of the graph embedding space enables the representation of more intricate relationships between nodes, while a smaller embedding space provides regularization, mitigating overfitting. Table 5.5 displays the accuracy achieved with different graph embedding space dimensions. In the case of smaller datasets like Cora and CiteSeer, the highest accuracy is attained with a dimension of 6 or 8, whereas PubMed necessitates a larger embedding dimension of 8.

Consequently, the optimal value for the embedding space dimension relies on the complexity of the dataset at hand, and the over simplification of $d = 2$ leads to lower accuracy.

Table 5.5: Performance reported with different graph embedding space dimensions d. The mean and standard deviation of accuracy on 10 runs are reported.

d	**Cora**	**Citeseer**	**Pubmed**
2	67.7 ± 6.7	63.08 ± 1.84	85.48 ± 1.34
4	75.5 ± 15.14	71.42 ± 2.1	87.72 ± 0.67
6	80.0 ± 0.94	**72.6 ± 1.25**	83.16 ± 14.81
8	**82.08 ± 0.86**	66.58 ± 18.88	**88.14 ± 1.27**

5.3.2.4 Graph embedding function f

The dDGM framework offers a range of choices for graph embedding function f: MLP, GCN and GAT. GCN and GAT functions use both the node features and the input graph connectivity. The MLP disregards the connectivity of the input graph and instead applies a multilayer perceptron individually to each node using the input features. In order to maintain parameter parity, the MLP is designed with a linear layer that matches the input and output dimensions of the convolutional operations. This linear layer is subsequently followed by a ReLU activation function.

The results demonstrate the important role of utilizing the input connectivity graph within the DGM module to achieve satisfactory performance, particularly for the Cora and CiteSeer datasets. Specifically, when the input graph connectivity is discarded by constructing a new graph solely based on input feature similarities (k-nearest neighbors), or when no connectivity information is employed (MLP), a significant decline in accuracy $(20 - 30\%)$ is observed. This signifies that in these datasets, the graph structure itself holds significant information regarding the node classification, emphasizing the importance of incorporating the graph structure into the learning process.

Table 5.6: Performance for dDGM reported with different graph embedding functions f. The mean and standard deviation of accuracy on 10 runs are reported.

	Cora	**Citeseer**	**Pubmed**
MLP	58.04 ± 9.07	50.84 ± 20.00	**87.44 ± 1.29**
GCN	**81.52 ± 1.76**	70.48 ± 1.12	86.98 ± 1.16
GAT	81.28 ± 1.68	**71.9 ± 1.37**	87.00 ± 1.14

6. Conclusions and Future work

In this thesis, we have achieved our main aims of reviewing the current graph structure learning (GSL) methods, empirically validating these methods through accuracy scores on citation network datasets, and analyzing the mechanism of these approaches along with the influence of hyperparameters, specifically in the case of discrete-DGM.

Through the empirical comparisons between the tradional neural network - Multilayer Perceptron framework and graph neural network based - GSL frameworks, namely Graph Convolutional Network, Graph Attention Network, and discrete-DGM, on citation network datasets of CiteSeer, Cora and PubMed, we have consistently observed that the graph structure learning frameworks outperform the Multilayer Perceptron in terms of accuracy for node classification tasks. Notably, the discrete-DGM framework, with its unique approach to learning underlying structures, has shown tremendous potential, especially when hyperparameters are carefully adjusted based on the dataset size.

Moreover, through the empirical results, it demonstrates that graph-based neural networks outperform traditional neural networks (Multilayer Perceptron) when working with graph-based datasets. It presents an evolutionary progression of Graph Structure Learning techniques, in the chronical order of presented frameworks, including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and Deep Graph Generative Models (dDGM), which successively improve validity and effectiveness. This represents a paradigm shift in handling graph data, leading to enhanced performance and validated results.

We have successfully addressed the following research questions in this thesis:

- How do the current state-of-the-art graph structure learning methods work, and how can we apply them to predict the class of a paper in a citation network?
- What is the level of accuracy achieved by different methods?
- How do changes in hyperparameters affect the behavior of the models?

In our comparative analysis, we have evaluated four frameworks: MLP, GCN, GAT, and d-DGM, for classifying the papers in a citation network. Moving forward, there are several avenues for further work in this area. For instance, exploring different techniques of graph structure learning, such as learning discrete graph structures [60] or considering c-DGM [61], could provide interesting avenues for comparison.

Expanding the scope to medical classification, particularly working with medical images where each node feature represents a combination of demographic features and medical images, could be a promising direction. The graph structures learned from graph structure learning methods can potentially reveal relationships between patients in a cohort, as mentioned in a related paper [62].

Additionally, investigating the models' behavior under adversarial attacks raises interesting questions:

- **Question 1:** How does reducing the message passing in graph neural network-based models, or trimming it off with different thresholds, impact the model's performance?
- **Question 2:** What is the influence of the learning rate on the models' performance?
- **Question 3:** How do the models perform in classifying a node with a certain number of neighbors? In other words, what is the accuracy of node classification based on its degree?

These aforementioned questions present potential areas for improvement and extension of graph structure learning methods, which can be explored as future work. A particularly intriguing research direction would be leveraging the learned graph structure to analyze model interpretability comprehensively. Furthermore, learning the edge weights would enable us to tackle a wide variety of problems, such as protein interaction analysis in the biological domain, as in a Nature paper [63].

In conclusion, this thesis has successfully achieved its objectives by evaluating and comparing various graph structure learning methods on citation network datasets. The results have shown the superiority of graph structure learning frameworks over the Multilayer Perceptron and highlighted the potential of discrete-DGM. The future work can focus on further advancements in graph structure learning techniques, exploring applications in medical classification, investigating models' behavior under adversarial attacks, and enhancing model interpretability through the analysis of learned graph structures.

7. Reproducibility

In this section, the datasets and results are described to ensure the reproducibility of the machine learning experiments. The code used for this thesis's experiments is publicly available on GitHub at: `https://github.com/lhthien09/validation_GraphStructureLearning/`. By providing open source code, the main aim is to enable others to replicate and build upon our work.

Bibliography

[1] Alexandru T Balaban. Applications of graph theory in chemistry. *Journal of chemical information and computer sciences*, 25(3):334–343, 1985.

[2] Sambor Guze. Graph theory approach to the vulnerability of transportation networks. *Algorithms*, 12(12):270, 2019.

[3] Nataša Pržulj. Protein-protein interactions: making sense of networks via graph-theoretic modeling. *Bioessays*, 33(2):115–123, 2011.

[4] Quoc Dinh Truong, Quoc Bao Truong, and Taoufiq Dkaki. Graph methods for social network analysis. In *Nature of Computation and Communication: Second International Conference, ICTCC 2016, Rach Gia, Vietnam, March 17-18, 2016, Revised Selected Papers 2*, pages 276–286. Springer, 2016.

[5] Shoujin Wang, Liang Hu, Yan Wang, Xiangnan He, Quan Z Sheng, Mehmet A Orgun, Longbing Cao, Francesco Ricci, and Philip S Yu. Graph learning based recommender systems: A review. *International Joint Conference on Artificial Intelligence*, 2021.

[6] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *Association for Computing Machinery - Computing Surveys*, 55(5):1–37, 2022.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Association for Computational Linguistics*, 2018.

[8] Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *The 2005 Institute of Electrical and Electronics Engineers/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 666–672. Institute of Electrical and Electronics Engineers, 2005.

[9] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Institute of Electrical and Electronics Engineers - transactions on neural networks*, 20(1):61–80, 2008.

[10] Samujjwal Ghosh, Subhadeep Maji, and Maunendra Sankar Desarkar. Graph neural network enhanced language models for efficient multilingual text classification. *14th International World Wide Web Conference*, 2022.

[11] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, Bo Long, et al. Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning*, 16(2):119–328, 2023.

[12] P Pradhyumna, GP Shreya, et al. Graph neural network (gnn) in image and video understanding using deep learning for computer vision applications. In *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 1183–1189. Institute of Electrical and Electronics Engineers, 2021.

[13] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1623–1625, 2022.

[14] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[15] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. Proceedings of Machine Learning research, 2017.

[16] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. Proceedings of Machine Learning Research, 2018.

[17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2018.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

[19] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 66–74, 2020.

[20] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9):e33, 2021.

[21] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[22] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[23] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.*, 2015.

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the Institute of Electrical and Electronics Engineers conference on computer vision and pattern recognition*, pages 779–788, 2016.

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[27] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[28] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *Institute of Electrical and Electronics Engineers - transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[31] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *The 2010 international joint conference on neural networks (IJCNN)*, pages 1–8. Institute of Electrical and Electronics Engineers, 2010.

[32] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations (ICLR)*, 2014.

[33] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[34] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[35] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *Institute of Electrical and Electronics Engineers - Transactions on Neural Networks*, 20(3):498–511, 2009.

[36] James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[37] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. Proceedings of Machine Learning Research, 2016.

[38] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[39] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[40] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *International Conference on Learning Representations 2018 Conference Blind Submission*, 2018.

[41] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. Proceedings of Machine Learning Research, 2018.

[42] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

[43] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*, pages 362–373. Springer, 2018.

[44] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *International Conference on Learning Representations*, 2018.

[45] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the Institute of Electrical and Electronics Engineers conference on computer vision and pattern recognition*, pages 5308–5317, 2016.

[46] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International conference on machine learning*, pages 2688–2697. Proceedings of Machine Learning Research, 2018.

[47] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1674, 2018.

[48] Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Xingcheng Fu, Cheng Ji, and S Yu Philip. Graph structure learning with variational information bottleneck. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4165–4174, 2022.

[49] Shaopeng Wei and Yu Zhao. Graph learning: A comprehensive survey and future directions. *arXiv preprint arXiv:2212.08966*, 2022.

[50] Xiaojie Guo, Yuanqi Du, Sivani Tadepalli, Liang Zhao, and Amarda Shehu. Generating tertiary protein structures via interpretable graph variational autoencoders. *Bioinformatics Advances*, 1(1):vbab036, 2021.

[51] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[52] Tianfan Fu, Wenhao Gao, Cao Xiao, Jacob Yasonik, Connor W Coley, and Jimeng Sun. Differentiable scaffolding tree for molecular optimization. *International Conference on Learning Representations*, 2022.

[53] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. A survey on graph structure learning: Progress and opportunities. *arXiv e-prints*, pages arXiv–2103, 2021.

[54] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[55] Jon L Bentley, Donald F Stanat, and E Hollins Williams Jr. The complexity of finding fixed-radius near neighbors. *Information processing letters*, 6(6):209–212, 1977.

[56] Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. Proceedings of Machine Learning Research, 2019.

[57] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *Conference on Uncertainty in Artificial Intelligence*, 2018.

[58] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. Watch your step: Learning node embeddings via graph attention. *Advances in neural information processing systems*, 31, 2018.

[59] Yu Chen and Lingfei Wu. Graph neural networks: Graph structure learning. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 297–321. Springer Singapore, Singapore, 2022.

[60] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. Proceedings of Machine Learning Research, 2019.

[61] Anees Kazi, Luca Cosmo, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael M Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *Institute of Electrical and Electronics Engineers - Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1606–1617, 2022.

[62] Kamilia Mullakaeva, Luca Cosmo, Anees Kazi, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael M Bronstein. Graph-in-graph (gig): Learning interpretable latent graphs in non-euclidean domain for biological and healthcare applications. *Medical Image Analysis*, 2023.

[63] K. Jha, S. Saha, and H. Singh. Prediction of protein–protein interaction using graph neural networks. *Sci Rep 12, 8360*, 2022.

List of Figures

List of tables