Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

An Investigation Into Graph Neural Networks

Vishal Kumar
[redacted]

Supervised by: Professor Rozenn Dahyot

September 2020

A Dissertation report
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science (Intelligent Systems)
School of Computer Science & Statistics
Trinity College Dublin, Ireland

Declaration

[redacted]

Vishal Kumar

September 7, 2020

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Rozenn Dahyot, who proposed this work and has not only provided immense support, guidance, and feedback through each phase of this dissertation but has also been a constant inspiration as a researcher.

I am grateful to Trinity College Dublin and the School of Computer Science and Statistics for helping me build a solid foundation of computer science principles as well as research methods and providing me with a platform to excel.

Also, I would like to thank course director Dr.John Dingliana who made things easy going with the course during the pandemic of COVID-19.

Lastly, I would like to thanks my family and friends, the moral support and motivation extended to me by them have been indispensable factors in the successful completion of this dissertation, which also kept me up while low times, and for that, I am truly thankful.

Any omission of acknowledgement does not reflect the lack of regard or appreciation.

Abstract

Graphs are a powerful representation of data that is ubiquitous and can be flexible in nature. There are many real-time applications where graphs are being used, such as recommending medication, protein interface prediction, handling traffic networks. However, these application areas make use of non-Euclidean graphical data, which involves highly relational or mostly dependent elements. They cannot be processed well by traditional machine learning approaches or deep learning models (e.g., CNN, LSTM, RNN). Most unsupervised learning methods (e.g., network embedding) cannot utilize the inherent logic contained in graph nodes. Inspired from deep learning architectures, graph neural networks (GNNs) are capable of conflating feature information from nodes and graphical structures to learn graph representation via feature propagation and aggregation. In this work, a concise introduction to basic graph concepts, GNN models, and their applications are provided. It starts with the introduction to the basic GNN models. Then several general graph frameworks and some widely used variants of GNN models are explained, such as graph convolutional networks, graph attention networks, graph recurrent networks. Application of GNNs is categorized based on structured and non-structured data, and then GCN models are designed using different graph libraries and trained on some datasets for solving classification tasks. With the analysis of intermediate and final results, it is observed GCN models can successfully process graph data and also outperforms traditional fully connected networks by 4% of classification accuracy. So it has proven that due to high interpretability, new architectures and libraries, and performance, there is a dramatic increase in applications and research of GNNs as a graphical analysis tool.

Keywords: DNN, GNN, CNN, graph, Graph Data, Deep Learning, Graph Neural Network, Graph Convolutional Network

Word Count: 14876

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Over the year, promising progress has been made by deep learning methods in several fields such as recommender systems, computer vision, traffic networks, physical systems, knowledge graphs, and natural language processing. In these fields, data are usually represented in the Euclidean domain. However, there exist several learning tasks that need to deal with non-Euclidean graphical data, which includes highly cohesive information elements, e.g., physical systems modeling, analyzing molecular fingerprints, protein interface prediction, etc. Graph neural networks GNNs are primarily inspired by deep learning architectures operating on graph structures. Many of GNN architectures are similar or comparable to popularly used deep neural net counterparts. Due to irrefutable performance and easy understanding, GNNs have become widely applied graphical analysis method.

A Graph data structure(sometimes called a network) highlights the relationships between different components in the data models. It consists of a set of nodes as objects, and their relationships called edges, which act as the connection between nodes. Such data models are beneficial to use when dealing with objects that have multiple links. Graphs are age-old data structures being used for centuries and widely used in applications related to current real-life scenarios including, LinkedIn, Instagram, and Facebook social networks, and route navigation applications such as Google maps using the road and street networks.

In recent years, there are many pieces of research made in the analysis of graphs using machine learning methods. Because of the natural way of information representation and a high expressive power every day, this data model is getting more and more attention. For instance, beyond immediate denotation, the Graphs exhibit a connotative ability to represent a large number of systems in several areas such as prediction of the missing piece of data in knowledge graph (Hamaguchi et al., 2018), design and discovery of drugs from protein interaction graph network (Fout et al., 2017), node embedding generation for unseen part of social network data represented as large graphs (William L Hamilton, J. Zhang, et al., 2017), interacting and understanding physical world structure based on graph networks (Sanchez-Gonzalez et al., 2018), convolution-based semi-supervised learning on graphical data (Kipf and Welling, 2016), modeling object interaction in complex physical systems (Battaglia, Pascanu, et al., 2016), reinforcement learning for NP-hard algorithms in combination with graph embedding (Dai et al., 2017). In essence, it can be construed that being a unique type of non-Euclidean form of data structure utilized in machine learning models, the understanding graph becomes necessary for analysis that focuses mainly on the classification of nodes, clustering data points, and relationship prediction among nodes.

Inspired by deep learning methods, Graph neural networks (GNNs) have been in existence for around 20 years. Still, in the last five years, attention towards this neural network has dramatically increased due to computational support by high-end hardware and GPUs. In this time frame, plenty of new architectures have emerged, novel areas of application found, new platforms to support computational needs have been implemented, and many dynamic libraries, especially to handle graph data, have entered into the scene. These include neural networks from vanilla graph neural network to advanced architectures such as graph convolution-based network(GCN), Graph Encoders and Decoders, Recurrent neural network architecture(RNN) based Graph Long short-term memory(LSTM). Libraries like Pytorch Geometric, Graph Nets, Deep Graph provide an efficient environment to model a graph neural network at ease. Highly scalable and robust GNN models help in modeling dependency and relationship among nodes in large graph networks, which enabled the significant breakthrough in research work related to graphical analysis of applications.

1.2 Motivation

In most cases, deep neural networks usually perform better than traditional machine learning methods. But when tabular data is considered; clearly gradient boosting-based machine learning models outperform fully connected neural networks(FCNN). Also, in the case of non-linear transformation that is considered as a unique characteristic of deep neural networks, deep neural networks(DNNs) can't be ranked above traditional Machine learning models. Many ML models can efficiently handle non-linearity in data structures, too, such as decision trees and SVM. It is rare to find a plain fully connected neural network has achieved state-of-the-art performance from the perspective of any particular benchmark. Based on applications, some specific neural network layers are responsible for setting DNNs apart from the result reached by traditional ML models. In a DNN model, these particular neuron layers allow parameter-sharing, such as the convolutional layer for space and recurrent layer for time. DNN advances well in extracting features automatically by utilizing these specialized neuron layers and effectively replaces the hand-engineered method of feature selection. By exploiting invariance in spatial translational of data, the convolutional layer in models such as Conv1D, Conv2D, and Conv3D works excellent for text, images, and 3D-images respectively. In contrast, DNN models such as GRU and LSTM having recurrent layers utilize temporal dependence in data and are suitable for speech and text processing.

Data types such as Social, Biological, and Knowledge have underlying graph structures (Z. Wang, 2020). E.g:

Social: citation network as well as social media networks such as LinkedIn, Facebook and Twitter, etc.

Biological: based on physical or regulatory interactions, organization of genes, and protein as graphs such as gene regulatory networks and protein-protein interaction networks, respectively.

Knowledge: organization of knowledge into graphs such as connection graph structures for Wikipedia articles and Google's knowledge graph for better information retrieval.

As the initial motivation, CNNs have the unique capability to extract and compose scalable spatial features with high representational power. This capability resulted in a revolution in deep learning methods and breakthroughs in most of the traditional machine learning models. Upon depth analysis of CNNs with graphs, critical features of CNNs are found as localized connection, weight sharing, and multi-layer usage (LeCun, Yoshua Bengio, and G. Hinton, 2015). These features enabled to solve many problems in graph domain because of:

- localized connection in graphs structure
- when compared to traditional spectral graph methods, sharing of weights dramatically reduces computational cost (Chung, 1996).
- multi-layer structure helps to handle hierarchical patterns enabling to capture multiple sizes of features.

However, CNNs applications area involves only regular Euclidean information such as texts(1D sequence) and images(2D grids), and interestingly these application areas can also be identified as instances of graphical structure data. As depicted in Figure 1.1, the transformation of CNN architecture from Euclidean to non-Euclidean application areas is hindered by the inability to present localized convolutional filters with pooling operators.

Graph embedding is a method to represent graph entities such as nodes, edges, or subgraphs in reduced dimensional vectors (Goyal and Ferrara, 2018; William L. Hamilton, Ying, and Leskovec, 2018; D. Zhang et al., 2018) acts as another motivation for this thesis. For graph analysis, traditional machine learning methods are dependent on manual feature engineering, and their performance is restricted by higher cost and inflexibility.

Inspired by the success of word embedding (Mikolov, Chen, et al., 2013), Deepwalk (Perozzi, Al-Rfou, and Skiena, 2014) is a learning method for latent representation vertices in a graphical network and is considered as first graph embedding method which utilizes

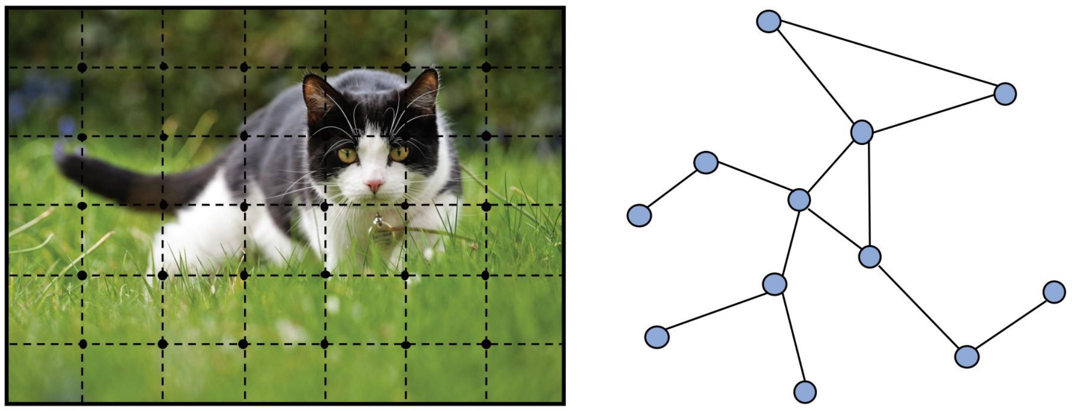


Figure 1.1: Image and Graph structure in non-Euclidean space

representational learning with SkipGram model (Mikolov, Chen, et al., 2013) on created random walks. Some other algorithms, such as node2vec(Grover and Leskovec, 2016) , LINE(Tang et al., 2015), and TADW(Yang et al., 2015), also achieved benchmark results. Irrespective of breakthroughs, these models have two significant drawbacks (William L. Hamilton, Ying, and Leskovec, 2018). First, the capability of generalization is not available in direct embedding methods, i.e., can't handle dynamic graphs and can't generalize to new graphs. Second, the encoder present in these models does not share parameters between nodes leading to inefficient computations, i.e., the linear relation between the number of nodes and the number of parameters.

Primarily, Graph neural networks are inspired by convolutional neural networks(CNNs) (Lecun et al., 1998). GNNs are a relationship-based model that captures graphs dependence via a method called 'message passing' between graph nodes, i.e., GNNs are an extension to the deep neural network to handle graphical representations of information.

Given the ubiquity of graphs in real-life applications and ongoing research work to analyze the effectiveness of Graph structure-based deep learning methods, this dissertation attempts to investigate state-of-the-art for deep learning methods on graphs and their variation based on the type of graph data. Also, some research questions are addressed, such as:

- How to handle data structures that do not have temporal or spatial structures instead have a graph structure?

- What kind of information is required to process a graph structure?
- What are the available library and frameworks to design a Graph neural network and how to implement those?
- How to process non-structured graph data using graph-based deep learning methods?
- How much efficient a GNN model is as compared to traditional machine learning models?

1.3 Dissertation Structure

This dissertation is organized as follows:

- The **first chapter** highlights the overview of the research work, introduces the research questions, problems addressed, motivations for exploring this research area.
- The **second chapter** details the underlying concepts that are used throughout this study, as well as state-of-the-art solutions for the problem identified and their comprehensive literature review.
- The **third chapter** presents the design of selected architectures, introduces the metrics for performance evaluation, frameworks, libraries used, and overall design workflow.
- The **fourth chapter** incorporates implementation methodologies, software and hardware setups, with a discussion on data and methods.
- The **fifth chapter** contains analysis and discussion of results obtained for different configurations and selected deep learning methods.
- The **sixth chapter** concludes the overall work, discusses the usefulness of achieved results, challenges encountered, highlights limitations of work and methods, and suggests improvements with possible future actions of the study.

Figure 1.2 shows the Gantt chart for dissertation timeline:

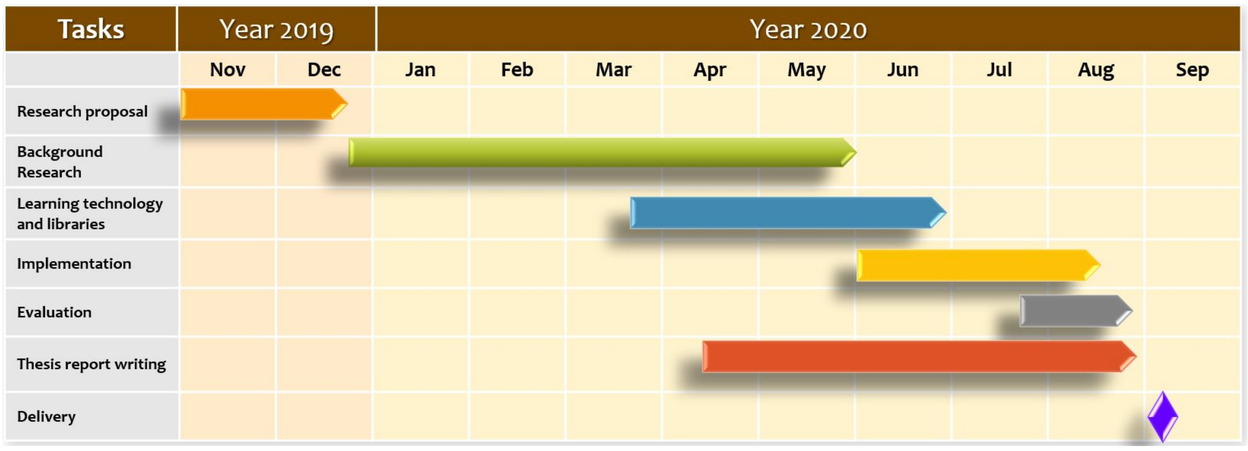


Figure 1.2: Dissertation Timeline

Chapter 2

State of the Art

2.1 Neurons and Neural Networks

In traditional machine learning models, the neural network holds a position that is compared to most of the benchmark algorithms. A simple neural network contains several connected neurons. Logically, these neurons bear a high resemblance to natural biological neural networks. A black box architecture is shown in Figure 2.1. While training, a typical neural network in the machine learning domain tries to learn from given inputs. It starts with random values (called weights); these weights are updated continuously by connections between neurons using the most commonly used algorithm (Wikipedia Contributors, 2019) called "back-propagation", which is based on the optimization of parameters using the gradient descent method. The back-propagation algorithm consists of two steps: "Forward calculation" to compute the value of weight for each neuron in forwarding order and "Backward propagation" to evaluate error for each variable and update the internal state of parameters in backward order using partial derivatives. This process of updating the internal state is repeated until precise performance is achieved in output, and learned weight values are stored for connections between neurons.

In recent years, many new architectures have emerged in the field of deep learning to represent a variety of neural network models. To improve performance and achieve generalization, most of the researches are focused on different learning algorithms and model architectures. These architectures are grouped into four categories as:

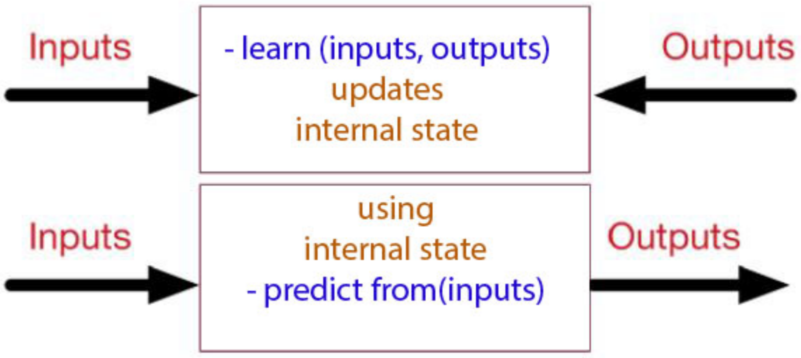


Figure 2.1: Black box view of the neural network

- **Feed-forward neural network:** In artificial neural networks, FNN, as shown in Figure 2.2, is the first and simplest architecture. FNN architectures comprise an input layer, some hidden layers, and final output layers (Sazli, 2006). There are multiple neurons in a layer, and each layer is connected to only neighbour layers. Based on the number of hidden layers, it can be called as single-layered or multi-layered FNN.

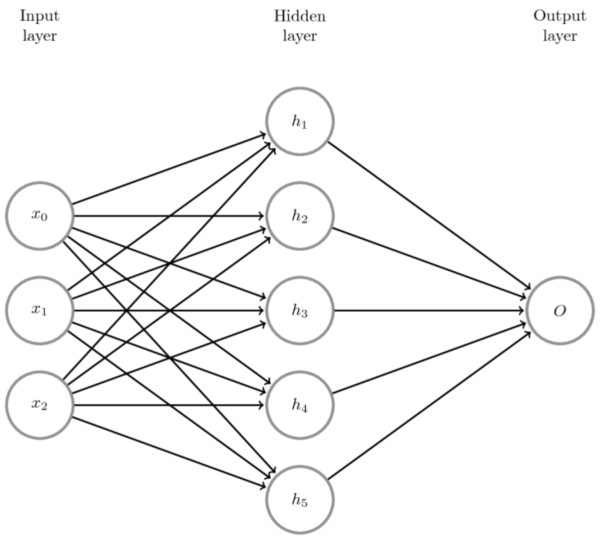


Figure 2.2: Multi-layered FNN (Sazli, 2006)

- **Convolutional neural network:** CNN is a particular type of Feed-forward neural network. Unlike FNNs, local connectivity is preserved in convolutional neural networks. Similar to multi-layered FNN, a typical CNN structure has convolutional layers, pooling layers, and many fully connected layers. It is one of the most researched deep learning models and used widely in applications related to Computer vision. CNNs have many classical architectures such as AlexNet (Krizhevsky, Sutskever, and G. E. Hinton, 2012) (Figure 2.3), VGG (Simonyan and Zisserman, 2014), and GoogLeNet (Szegedy et al., 2015).

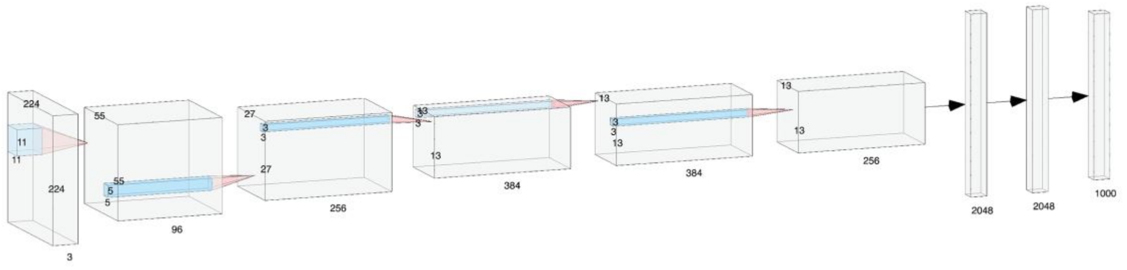


Figure 2.3: Model architecture for Alexnet network

- **Recurrent neural network:** RNN is developed to handle sequential or series of data. Similar to FNNs, Recurrent neural architecture receives input from other neurons, but a unique memory mechanism allows it to maintain a piece of historical information about processed input data. This memory mechanism enables RNN to handle continuous series data efficiently. A problem called "long-term dependency" (Y. Bengio, Simard, and Frasconi, 1994; Hochreiter et al., 2001) can occur in basic RNN models, so some variants of models are researched which introduce the gate mechanism called GRU (Cho, Merriënboer, et al., 2014). Since RNNs are useful to process series data, so they are mostly used for text and audio analysis.
- **Graph neural network:** GNNs are specifically designed to handle graphical data or non-Euclidean data structures. Detailed researches with variants of models and Applications are discussed in Section 2.3 and 2.4, respectively.

2.2 Geometric Deep Learning

Some research fields like social science, communications, genetics, and computer graphics analyze data with non-Euclidean in nature, such as social network, sensor network, regulatory network, and mesh-shaped surface, respectively. Geometric deep learning (GDL) is a specific type of deep learning method which has recently become one of the foremost researched topics in machine learning. GDL is termed as a group of emerging techniques to generalize deep learning models of structured or Euclidean domains to graphs or manifolds in non-Euclidean domains (Bronstein et al., 2017).

2.2.1 Deep Learning Models for Manifolds

By definition, Manifolds are spaces which are locally euclidean in nature. The most accessible and straightforward example is modelling the spherical surface of planet Earth, which can be considered flat in a local region. As per(Wilson, 2012) “a differentiable n-dimensional manifold X is a topological space named M where each point x has a neighbourhood, i.e. $x \in M$ is homeomorphic to n-dimensional euclidean space or R^n, also called tangent space”. In the research community of computer graphics, several efforts are made to generalize deep learning models to manifolds; mainly, 3D surfaced models where these three-dimensional objects are commonly referred to as “Riemannian manifolds”.

(Masci et al., 2015) utilizes geodesic polar coordinates to apply local patch filters and introduces an intrinsic type of CNN for manifolds that can achieve state-of-the-art performance for applications as description and retrieval of shapes. In the similar application area, (Boscaini et al., 2016) employed anisotropic heat kernels (Andreux et al., 2014) as the alternative to retrieve integral patches in manifolds. (Sinha, Bai, and Ramani, 2016) proposes the method to obtain the Euclidean model of 3D surfaces using geometric image representation and shows the application of standard CNN on such 3D shapes. These spatial techniques can be generalized among different domains. Still, the construction of spatial models such as heat kernels uses a unique geometrical representation for manifolds, which cannot be generalized for graphs.

2.2.2 Deep Learning Models for Graphs

Graphs are a simplified model of data construction for networks, similarities, interaction, and dependencies between different objects. Initial work to generalize neural networks for graph domains is found in (Gori, Monfardini, and Scarselli, 2005; Scarselli et al., 2009), which is inspired from recurrent neural networks and includes most of the types of graphs as directed, undirected, cyclic and labeled. Due to computation complexity, these works remain unnoticed until, (Yujia Li et al., 2015) introduced optimized feature learning techniques for graph-structured data with gated recurrent blocks. Similar to CNN (Bruna et al., 2013; Henaff, Bruna, and LeCun, 2015) proposed deep neural architecture for the graph in the spectral domain, using inference among classical Fourier transforms and projections on eigenbasis for graph Laplacian operator. This work created a surge in interest and research works for in-depth analysis of the non-Euclidean models, especially in domains like computer vision and traditional machine learning models. Following this model,(Defferrard, Bresson, and Vandergheynst, 2016) introduce a fast and localized filtering technique independent of computing eigenvectors explicitly and uses "recurrent Chebyshev polynomials". This approach is simplified in (Kipf and Welling, 2016) by using simple filters that 1-hop neighbourhood techniques on graphs nodes. In the network analysis domain, (Mikolov, Sutskever, et al., 2013) proposes the Word2vec technique, which inspired many types of research as (Tang et al., 2015) to generate graph embedding methods.

Since Fourier basis or Laplacian eigenbasis is dependent on the domain of applications, and convolution in the spectral-domain is dependent on Fourier basis, which implies that CNN model with spectral approaches such as (Bruna et al., 2013; Defferrard, Bresson, and Vandergheynst, 2016; Henaff, Bruna, and LeCun, 2015) can learn on one graph, but, this learning cannot be generalized to another graph with separate Laplacian eigenbasis.

2.3 Graph Neural Network Variants

The initial graph neural network proposed by (Scarselli et al., 2009) is considered as the original GNN, which had labeled nodes for information, undirected edges. It works on the most straightforward graph format. Many forms of graphs neural networks have emerged based on input graph types, which enhance the representational ability of the original model, modifications in propagation steps (such as convolution, gate, or attention mechanism) for learning graph representation with better quality, and improving training efficiency.

This section highlights the researches available for GNN models in categories, i.e., Sec 2.3.2 lists models operating on individual graph types, Sec 2.3.3 focuses on modified GNN models based on propagation steps, and Sec 2.4 explains models utilizing advanced training methods.

2.3.1 Basic Models

Deep learning methods have reached breakthroughs in many applications areas, most notably in computer vision, speech and text recognition, and natural language processing. 1D, 2D, or 3D Euclidean-structured data such as audio signals, video streams, or images were in focus for most of the traditional deep learning architectures. In recent years, several attempts are made to generalize deep learning architectures for non-Euclidean structured data such as graphs and manifolds. Researchers have investigated to extend CNNs from grid-structured data to generic Non-Euclidean structured data. Non-Euclidean structured data have a variety of applications from domains of computer graphics, modeling physical systems, or network analysis. Start:

(Boscaini et al., 2016) proposes Anisotropic Convolutional Neural Network to generalize classical CNNs to non-Euclidean. ACNN replaces the traditional convolution process by projections over a group of anisotropic diffusion kernels. ACNN can learn the complex correspondence between irregular shapes, which is the fundamental problem in applications requiring graph processing; however, this framework lacks to learn graphical structured non-Euclidean data.

Based on local geodesic charting procedures in polar coordinates to extract patches, (Masci et al., 2015) introduces a generalized CNN framework for two-dimensional non-Euclidean manifolds as Geodesic Convolutional Neural Networks (GCNN) to learn task-specific features in applications as shape retrieval. Then, these patches are processed through layers of filters and linear or non-linear operations. Filter coefficient and weights of linear combinations that act as optimization factors are learned to reduce the cost for specific tasks. Since GCNN uses built-in spectral features as input to the network, it can't be defined on generalized functions defined on manifolds.

(Bronstein et al., 2017) Geometric deep learning architectures can be considered as an umbrella term for emerging architectures to generalize structured deep learning methods to non-Euclidean domains. This work provides a detailed overview of geometric deep learning problems and their solutions, areas of applications and difficulties, and direction for future work. It suggests generating spectral convolution may help to design CNNs for a specific graph, but the same model can't be applied to another graph. It is found that generalization of the model across domains is also necessary for generalization of deep learning methods to geometric data, only designing non-Euclidean counterparts for basic building blocks(convolutional and pooling layers) are not sufficient. Besides, the dynamic nature and direction of the relationship in graph data should also be kept in focus.

While many pieces of research focus on generalizing convolutions for graphs and manifolds, this thesis work is only focussed on methods defined on graphs with the investigation of other variants and applications of GNNs. In deep learning models, (Xiaolong Wang et al., 2017) introduces convolutional and recurrent operations which act as building blocks to work on local neighborhood one-by-one. The non-local means (Morel, Coll, and Buades, 2005) is a filtering algorithm based on non-locally weighted averaging of every pixel in an image. It allows all pixels at a long-range to participate in filtering response for a local neighborhood based on the similarity of the patch used.

(Boaz Lee et al., 2019) produces a continuous and focused work on graph mining that has demonstrated several useful insights from structured graphical data. In real-world applications, a graph can be both large in size and noise in nature; these noises can act as the obstacle for practical graph mining. Similar to the attention model in deep learning methods for Euclidean data, an efficient way to deal with noise in the graph is to incorporate the "attention" mechanism into learning models. In graph setting, attention mechanisms can help a method to focus on a task-relevant part and generate better predictions. Attention can be used to find the importance of features from neighbors and assign them to target nodes. In Fig: 3, the size of the edge defines weight in the attention given to neighbor nodes. E.g., by focusing on target nodes classmates, the activity type of target node can be predicted.

(Boaz Lee et al., 2019) categorizes existing work for graph attention models into three groups, which are based on the method of problem settings(the type of input and output),

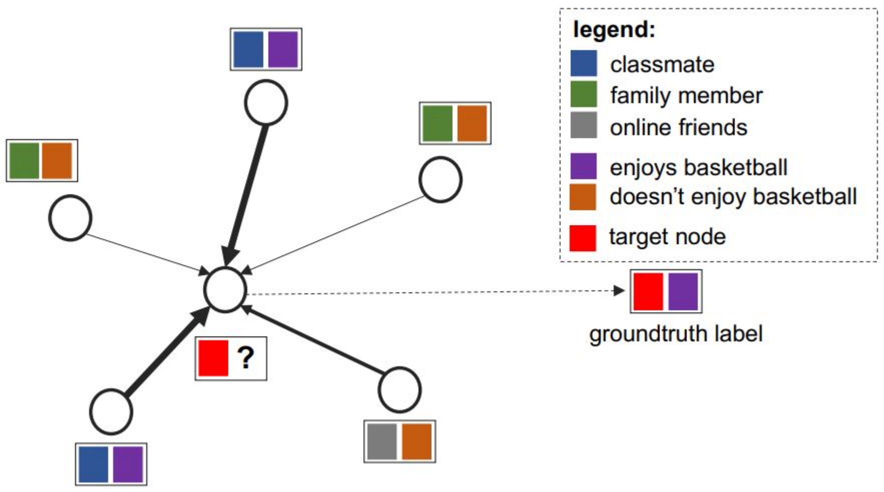


Figure 2.4: Attention given to neighbor nodes

attention type(similarity or weight-based) and type of task(node/edge or graph level)

Intensive surveys for neural graph networks are done in (Y. Zhang et al., 2018) and (Wu et al., 2020), which are focused on discussing a variety of GNN models. (Y. Zhang et al., 2018) explore advances in the graph analysis method and reviews different variants of deep learning methods on graphs. (Wu et al., 2020) discuses an overview of GNN in the machine learning domain and data mining. It also proposes taxonomies to categorize state-of-the-art graph neural networks.

2.3.2 Graph Types

There are many varieties of graphs that model the unique forms and need specific GNN architectures. Some graph neural networks modeled for particular types of graphs are highlighted here:

Directed Graph:

The directed graph is the first of many kinds of graphs. Logically a graph with undirected edges can be treated as a graph with a two-directional edge showing the flow of information in both ways from each end node. But, practically directed graph representation more dense information as compared to undirected graphs. E.g., in the domain of knowledge graphs, directed edges contain a head node and tail node for edges, which are known as parent class and child class, respectively. While propagation of information, parent and child classes are processed separately. When graph size becomes more significant, distribution of information from distant nodes can suffer from the problem of information dilution (Bruna et al., 2013) introduces a Dense Graph propagation to avoid this problem by designing direct communication links for distant nodes. DGP utilizes the hierarchically structured property of knowledge graphs with the direct link for nodes based on the type of relationship with other nodes such as ancestor or descendent. DGP also uses a weighting method where the two-weight matrix is defined as one for the parent and other for the child node; these weights contribute depending on nodes distance.

Heterogeneous graphs:

It is another essential type of graph where many different types of nodes are present. Usually, heterogeneous graphs are handled by modifying actual node features, i.e., appending the original feature node with a one-hot feature vector that is extracted from the specific feature of the same node. For information propagation (Y. Zhang et al., 2018) suggests " GraphInception ''- a classification process based on a deep convolutional network approach, which is aimed to learn node relationships in Heterogeneous Information Networks (HINs). GraphInception employs the meta path concept, where a group of neighbor nodes is created based on distance and node type. Each group is considered as a sub-graph (as presented in a homogeneous graph), information propagation is performed, results are appended for sub-graph propagation for combined node representation.

Similarly, heterogeneous graphs can also be processed based on hierarchical attention models, like (Xiao Wang et al., 2019) proposed a heterogeneous graph neural network (HGNN) that focuses on node and semantic level attentions in the graph. The node-level attention is meant for learning relation among nodes, and meta path generated using their neighbor nodes, whereas learning importance between individual meta path comes under the semantic level attention mechanism. After considering the importance of both semantic and node-level attention, HGNN forms node embedding by hierarchical aggregation of meta path-based neighbor nodes.

Dynamic graphs:

Data forecasting based on spatial-temporal information is essential and commonly applied in domains such as climate prediction and traffic detail forecasting. Many of such application areas involve prediction task by modeling on another variant of graph called a dynamic graph, that has static graphical data and dynamically changing signals as input data. Usually, in such scenarios, historical graph state is given, and future graph state is supposed to be predicted. In Figure 2.5 example of a spatial-temporal graph is shown where individual G_t represents a frame of current graph state at some time t.

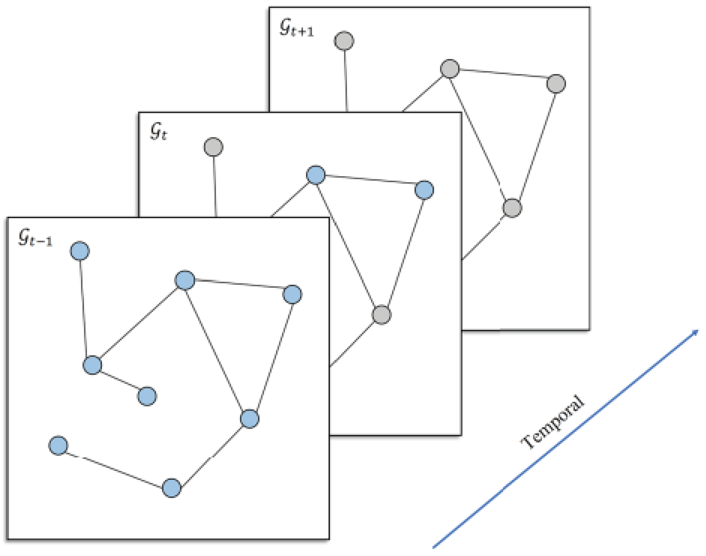


Figure 2.5: Example of a spatial-temporal graph.

Traffic forecasting involves challenges like spatial dependency and non-linear temporal characteristics of the road networks. These challenges are addressed in (Yaguang Li et al., 2017), as it represents the traffic flow as a diffusion method applied on a dynamic graph and proposes Diffusion Convolutional Recurrent Neural Network (DCRNN) architecture, which considers both spatial and temporal information. Spatial and temporal data are collected using GNNs, and the result is sent as input to sequence models such as CNNs. In contrast, (Jain et al., 2015) discusses an approach to utilize the strength of high level spatial and temporal information and sequential learning of recurrent neural networks (RNNs) all together at the same time. This method can be generalized for transformations of any spatial-temporal graph by performing some set of operations.

Graphs with edge information:

In graph models, edges can also contain useful information as weight or type of relationship between nodes. These kinds of graphs can be processed in two ways: first, modify the given graph to a bipartite graph where actual edges are represented as new nodes, and second, edge with information is split into two new edges. (Beck, Haffari, and Cohn, 2018) uses a new model to encode complete graphical structured details and allows nodes and edges to exhibit implicit feature representation. Using a similar type of graph with edge information (Schlichtkrull et al., 2017) introduces Relational Graph Convolutional Networks (R-GCNs). It employs it to tasks of Link prediction and classification of nodes. Link prediction involves tasks such as prediction of relationship or triples, i.e., subject-predicate-object. In contrast, the classification of nodes includes the prediction of node attributes. R-GCNs are specially designed to handle a high level of multi-relational data.

Multi-dimensional graphs:

Unlike all kinds of graphs (with binary edges) discussed earlier, graphs used in the real world applications involve multi-relationships among nodes that are known as the "multi-dimensional graph or multi-graph". Figure 2.6 shows an example of a multi-graph having a graph and dimensional expansion. For example, in web-based application Instagram, content sharing and user interaction can be either "comment", "like" and "share". In such cases applying a one-dimensional graph model may give an incorrect prediction because the relation type between nodes is not independent naturally. (Y. Ma et al., 2018) analyses this problem and introduces a multi-dimensional convolutional neural network model (mGCN) to handle a large set of data in learning collective node representation for multi-graph.

Recently, some unique kinds of multi-graph are designed such as (Khan and Blumenstock, 2019) develops graph-based convolution architecture to perform learning on a multi-view network. It uses two certain operations to convert multigraph into a single-dimensional graph, which starts from combining multi-view using sub-space analysis and then optimization of the graph by learning manifolds. Similarly,(Sun et al., 2018) constructs an algorithm called "multi-view network embedding (MVNE)" to reduce multi-view network

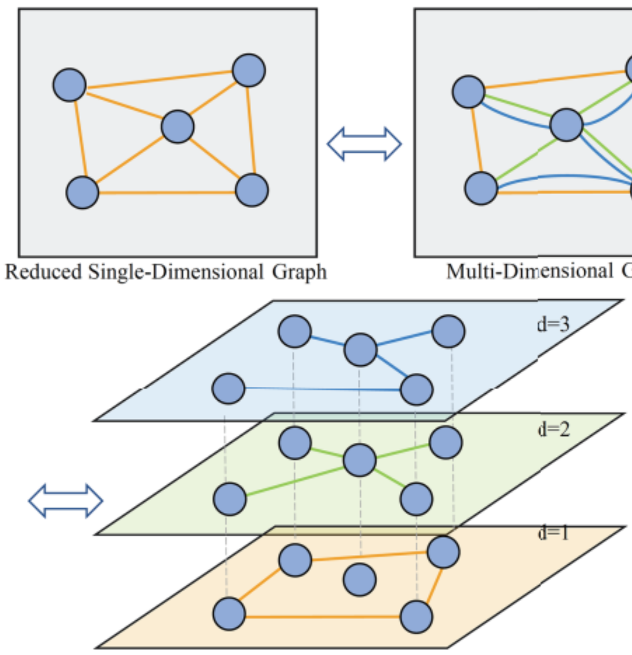


Figure 2.6: Example of multi-graph having a graph and dimensional expansion.

to lower-dimensional node embedding.

2.3.3 Propagation Steps

Designing propagation methods and output steps are an essential part of modelling a graph, as it helps to extract the implicit state of nodes or edges. Several pieces of research have emerged to enhance the propagation method, while for the output step, a simple feed-forward neural network is used. These researches use a unique aggregator to collect data from neighbours of nodes and different updater modules to modify the implicit state of nodes. This variety in propagation steps results in a variety of GNNs:

Convolution:

Many pieces of research in graph domains are focused on generalizing the convolution method to non-Euclidean domains. These works are grouped on approaches taken for kinds of graph representation as spectral and non-spectral.

"Graph Convolutional Neural Networks" are popularly used for graphical data representation and applied to learn semi-supervised applications. Unlike previous graph CNNs that do not apply to semi-supervised tasks and use fixed graphs, Graph Learning Convolutional Network(GLCN) is introduced in (Kipf and Welling, 2016) for representing graph data and semi-supervised learning tasks. GLCN unifies both graph learning methods and convolution in graphs. It also efficiently incorporates given and estimated labels to generate supervised data and utilizes them to construct the graph with convolution operation for estimation of the unknown tag. Being scalable in nature, GLCN outperforms traditional state-of-the-art fixed graph-structured CNNs; however, its memory requirement increases linearly with the size of the graph and does not handle the weight and edge direction between nodes.

In Non-spectral approaches, convolutions are generalized directly on the graph structure, and they work on neighbours that are spatially near. In this approach, designing convolution for random sized neighbours becomes a challenging task. Through diffusion-convolutional neural networks (DCNNs), (Atwood and Towsley, 2016) proposes the diffusion-based representations to learn graph-structured data and uses it as an efficient node classification method. DCNN exhibits qualities like isomorphic invariant graphical data representation, learning, and prediction for time-based polynomials. These qualities make DCNN perform better than probabilistic relational models (Koller and Friedman, 2012) and kernel-on-graph methods (Fouss et al., 2012), but DCNN lacks in scalability as it can't be applied to more massive graphs with millions of nodes; also, it does not possess the capability to encode spatial dependencies in the broader scope of nodes

(Monti et al., 2017) introduces a coordinated framework called Mixture model networks (MoNet), which follows the idea of spatial domain methods. MoNet framework allows generalizing CNN based deep learning architectures to handle non-Euclidean structured data such as graphs (Kipf and Welling, 2016) and few models on manifolds (Atwood and Towsley, 2016) by learning local, integrative task-specific, and stable features. MoNet outperforms traditional approaches on standard tasks, i.e., analyses of images, graphs, and 3D shapes.

Gate

Many new methods utilize gated mechanisms such as GRU (Cho, Merrienboer, et al., 2014) or LSTM (Sepp Hochreiter and Schmidhuber, 1997) for enhancing the propagation of data availability duration across the graph with diminished restrictions as compared to the original GNN model. (Zhuang and Q. Ma, 2018) presents a simple but scalable dual graph convolutional neural network (GGNN), which is designed to handle local and global consistencies for a graphical network. This model utilizes two convolutional networks and learning methods to collect both types of consistencies.

Attention

In the traditional machine learning domain, the attention mechanism is widely used for sequential task models such as machine reading (Cheng, Dong, and Lapata, 2016) and translation (Bahdanau, Cho, and Yoshua Bengio, 2015). Unlike the original GNN, which considers all neighbours equal in contribution, the attention models treat neighbours differently and assign separate attention score values to each neighbour. This assignment results in categorizing important and essential neighbours. Usually, these attention mechanisms are included in the propagation steps. (Veličković et al., 2017) introduces a “graph attention network” (GAT) that includes the attention mechanism while performing the propagation step. This model utilizes a self-attention method to collect implicit state information of a node by traversing neighbours. Self-attention method is implemented by employing a single attention layer.

Skip Connection

Some applications try to use more GNN layer together as stacking to get a better output, using the basic concept as more layers can help nodes to collect extra neighbourhood information. But, in reality, many applications can perform worse in the case of deeper models (Kipf and Welling, 2016), because expanded neighbours can cause noisy data, which can also be propagated through the increased number of layers. To handle this problem, (He et al., 2015) proposes a residual learning framework in the computer vision domain. This framework helps to train the deeper layer model by formulating layers as a residual function of learning.

2.4 Training Methods in Graph

Some advanced training methods are researched as original GNN has few drawbacks in training and step optimization. In particular, the GCN model calculates the complete graph Laplacian, which involves an expensive computational process in case of large graphs. Also, computing node embedding at layer L is a recursive process, and based in neighbours in layer 'L-1'. So if the number of layers (L) is increased, then the size of the receptive field of a node increases exponentially.

Sampling

For better effectiveness and scalability in GNN models, sampling techniques are added. (William L Hamilton, Ying, and Leskovec, 2017) suggests the GraphSAGE model, which tries to replace full graph Laplacian with the learning process in the aggregation method and can be generalized to unprocessed to nodes also. Initially, it aggregates the embedding of neighbourhood nodes, appends to node embedding of the target node, and concatenated information is propagated to the next layer in the model. GraphSAGE uses this learned aggregation and generates node embedding of unseen nodes too, i.e., instead of calculating embedding of individual nodes, an aggregation function is learned to generalize embedding by sampling features from neighbourhood nodes.

Hierarchical Pooling

In the deep learning model for computer vision applications, a pooling layer is required following a convolutional layer to capture generic features. Usually, large-scale complex graphs involve plenty of hierarchical information for node or graph level classification.

So many researches are done for designing layers for hierarchical pooling in graphs. (Simonovsky and Komodakis, 2017) tries to explore such implicit hierarchical features and introduces the Edge-Conditioned Convolution (ECC) design, where pooling layers are created by using the recursive downsampling method. The downsampling method splits the original graph into two-part using the sign of eigenvector with maximum size in Laplacian information.

2.5 General Graph Frameworks

Many graph frameworks are available to integrate different graph neural networks models into one framework, such as (Gilmer et al., 2017) suggests message passing neural network (MPNN) , which utilizes the basic building blocks structure of deep learning methods such as batching and normalization. MPNN framework is targeted to generalize GNN and GCN methods. (Xiaolong Wang et al., 2017) proposes another framework called as non-local neural network (NLNN) to aim computer vision tasks by generalizing 'self-attention' methods on graphs. Unifying both MPNN and NLNN, (Sanchez-Gonzalez et al., 2018) introduces the graph network (GN) framework, which also includes several other interaction and relational networks.

Message Passing Neural Networks

In the domain of chemistry (drug discovery and medicine science), several efficient and neural network models are present, which are invariant to symmetry in molecules. These models generate a function of the entire input graph by learning a message-passing algorithm with the aggregation process. In (Gilmer et al., 2017) a specific variant to this generic method is found and applied to chemical prediction to achieve benchmark results. A new generalized framework called Message Passing Neural Networks (MPNNs) is created by reformulating existing models. MPNNs is targeted to predict the quantum properties of organic molecules by modeling a computationally expensive calculation. MPNNs allows long-ranged interaction between nodes. Since it uses spatial information while training, it is challenging to generalize MPNNs to larger molecule size.

Non-Local Neural Networks

NLNN aims at capturing long distances relationships in deep neural networks by generalizing classical non-local mean operation used in computer vision applications. It calculates response at any position as the weighted sum of all neighbourhood position features. These neighbourhood position features can be in the domain of space, time or space and time both. The generated building block can be used in applications based on computer vision architectures.

Graph Networks

(Battaglia, Hamrick, et al., 2018) presents a framework called Graph Network (GN) to generalize and extend various approaches in neural networks used for the graph. GN is inspired by a flexible learning approach and utilizes high relational inductive biases to draw advantage from explicitly defined representation and computations. Graph networks use customizable graphs as building blocks to handle complex architectures, and their relational inductive biases, i.e., specific assumptions in design structures, promote better sample efficiency than traditional machine learning methods. However, graph networks exhibit a higher level of abstraction, and the classification of the application area is roughly defined.

GNN benchmarking framework

(Dwivedi et al., 2020) suggests a framework to benchmark GNNs, which identifies whether architecture types, initial principles, or methodologies incorporated can be generalized or upscaled when a larger and complex dataset is encountered. It has evaluated GNNs performance on medium-scale datasets, showcases the usefulness of graph networks, which are theoretically expressive and based on the basic message passing method.

2.6 Graph Applications

GNNs have been applied and tested on a wide range of application domains, such as learning based on supervised, unsupervised, and semi-supervised methods. Here, these applications are structured into two scenarios, structural and non-structural. And, some representative applications are summarized under both categories.

Structural Scenarios

It represents applications where the information contains an explicit relational structure such as modeling real-world physical systems, prediction of chemical properties based on molecular structures.

The relational structure of objects in real-world systems can be modeled as a graph using GNN based reasoning, which can represent a simpler view of physics, objects as nodes and relations as edges. (Battaglia, Pascanu, et al., 2016) introduces an interaction model for such physical systems, where the interaction of objects can be predicted, and implicit properties can be inferred. The major task of this model is to predict the new physical state of the system by utilizing relational object data, the implicit reason behind the interaction, and physical dynamics of objects.

The drug designing in chemistry with the aid of machine learning methods can also be considered in this scenario, where molecular fingerprints are determined using features of a particular molecule. Applying GNN to a molecular graph structure can result in better molecular fingerprint prediction. (Duvenaud et al., 2015) suggests a standard neural network model that can handle arbitrary graph information and generalizes extraction of the substructure feature vector by employing a method based on circular fingerprints.

Non-Structural Scenarios

Applications based on multi-agent systems, text, source code programming, and images are non-structural scenarios. Usually, there are two methods to apply GNNs on non-structural data- first, acquire structural properties from other domains to enhance performance; second, assume the relational information in the scenario and supply these inputs to the model designed for graphs.

The task of image classification in computer vision is a non-structural scenario where zero-shot and few-shot learning (X. Wang, Ye, and Gupta, 2018) methods are getting attention, and GNNs are used to incorporate structural information. The classification based on zero-short recognition can also use knowledge graph information. (X. Wang, Ye, and Gupta, 2018) builds a knowledge graph to predict the classifier on different categories of objects/nodes and suffers from over smoothing as it uses six GCN layers that remove important features too. (Kampffmeyer et al., 2019) tries to solve this problem by covering a larger neighborhood with both one and multi-hops on objects/nodes in the graph and only one GCN Layer.

Chapter 3

Model Designs and Libraries

3.1 Original Graph Neural Network

In traditional machine learning algorithms, the data used are mostly Euclidean in nature. As the initial step, Inputs are converted into numerical representation or features, and in many scenarios, statistical data is represented in the cartesian coordinate system. For deep learning approaches, and the embedding and flattening layers are employed to present any textual data and image grid data, where any implicit relationship among data is missed before the preprocessing layer only. In other words, sparse vector representations hold no meaningful relationship among inputs or miss comparative information. As a result, any similarity or distance between input feature vectors become the same. To overcome this problem, "distributed vector representation" is considered. In this data representation or feature creation method, generated features may or may not exhibit any useful relationship with input data but posses a comparative value denoting similar inputs that are encoded to the same features. For example, a vectorized representation of a dog will have similarity (in the sense of distance) or close relationships to other animals like cats, rather than any flower or non-living objects. But this similarity can't be found in "non-distributed vector representation".

The term Graph neural network is first coined in (Gori, Monfardini, and Scarselli, 2005) as a neural network approach to process graphs. Later (Scarselli et al., 2009) proposes a new neural network to process information in graph domains, which is considered as Vanilla GNN. It utilizes a function $\tau(G, n) \in IR^n$ to encode graph 'G' and each node from a non-euclidean into n-dimensional Euclidean space. After this, a supervised learning method is used in parameter estimation.

Graph-based applications can be grouped into two classes as graph and node focused tasks. In graph focused tasks, nodes do not influence function T, and it uses a regressor or classifier for graphical datasets such as modelling a chemical compound with atoms and chemical bonds between atoms. Whereas. In node-based tasks, function T and regressor or classifier also depend on each node. Tasks such as detection of objects in an image with local position estimation are node focused applications. The vanilla GNN is useful for both classes of graph applications.

3.1.1 Model

The vanilla GNN (Scarselli et al., 2009) can process graphs such as directed, undirected, and cyclic based on application design. The basic idea is that objects or concepts can be represented as nodes and their relationships as edges in a graph. And, each object is represented using its input features and link among other objects.
A simple graph 'G' is denoted as pair (N, E), where N and E represents a set of nodes and edges in the graph.
A node 'n' is naturally represented with its features and relationship to neighbour nodes using edges.
The set of nodes connected to node 'n' and set of edges with node 'n' as one of the vertices is denoted as ne[n] and co[n].
Each node and edges may have state or labels or input features denoted with real vectors; labels for node 'n' and edge (n_1, n_2) can be shown as l_n and $l_{(n_1,n_2)}$, where l is the vector that includes all labels of graph 'G'.
In Figure 3.1, the vanilla GNN model is applied for an undirected graph with 'l_n' and $l_{(n_1,n_2)}$ as input features or labels for node 'n' and edges with nodes n_1 and n_2.
The motive of vanilla GNN is learning the node or state embedding $x_n \in R^s$, which acts as encoded information based on neighbour nodes of 'n'. And, this state embedding x_n

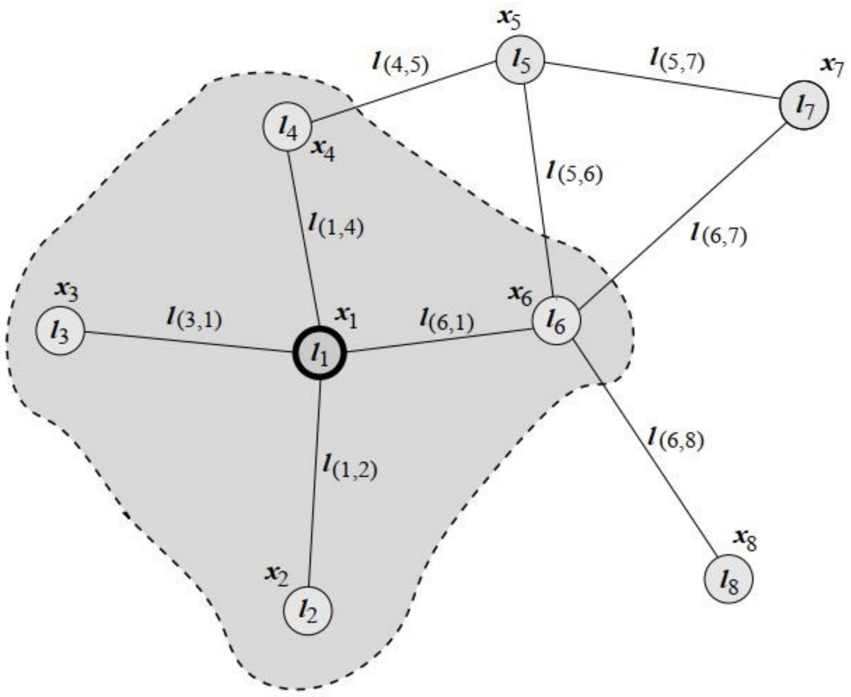


Figure 3.1: Example of the graph used in the vanilla GNN (Scarselli et al., 2009)

is useful to find output o_n, like the prediction of node label distribution.

Two parametric functions - 'local transition function' and 'local output function' are represented as 'f' and 'g'.

Parametric function 'f' is used to evaluate node state x_n as per input features of neighbourhood nodes. This function is used for each node in the graph. And function 'g' is used to find the output of node 'n'.

$$x_n = f(l_n, l_{co[n]}, x_{ne[n]}, l_{ne[n]}) \tag{3.1}$$

$$o_n = g(x_n, l_n)$$

Where 'l' is input feature vector, 'x' is hidden state, co[n] and ne[n] are set of edges and neighbourhood nodes of node 'n'. In other words,

l_n is input features of node n,

$l_{co[n]}$ is features of edges,

$x_{ne[n]}$ is node or state embedding neighbourhood nodes and

$l_{ne[n]}$ is the input features of neighbourhood nodes

In the example In Figure 3.1 for node ‘n=1’, l_1 is the input feature,
set co[1] includes $l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}$ as input features of edges connected to node ‘n=1’,
set $x_{ne[1]}$ has x_2, x_3, x_4, x_6 hidden state for neighbour nodes to node ‘n=1’,
And set ne[1] contains l_2, l_3, l_4, l_6 as labels or features of neighbour nodes of node ‘n=1’.
For example, state for node ‘n=1’ can be written as:

$$x_1 = f(l_1, l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}, x_2, x_3, x_4, x_6, l_2, l_3, l_4, l_6)$$

Upon stacking all edge and node features, state embedding with distribution output, matrices ‘X’, ‘O’, ‘L’, and ‘L_N’ are constructed as a compact representation.

$$X = F(X, L) \tag{3.2}$$

$$O = G(X, L_N)$$

Where ‘F’ and ‘G’ are called global transition and global output functions, respectively. Here, ‘F’ and ‘G‘ can be identified as stacking of all local ‘f’ and ‘g’ functions together.

For computing the next state in iteration, “Banach’s fixed point theorem” (Khamsi and Kirk, 2001) is used. GNN uses the simple iterative scheme as

$$X^{(t+1)} = F(X^t, L)$$

Where X^t represents t^{th} iteration for X and for any initial state X(0), this equation has exponential convergence for finding the solution.
When target information is available, for instance, t_n denotes target for specific node ‘n’; the loss can be calculated as

$$loss = \sum_{i=1}^{p} (t_i - o_i) \tag{3.3}$$

Where ‘p’ represents the count for supervised nodes

For learning parameters of local transition(f) and output(g) functions, gradient descent strategy is utilized as per steps:

1. Update node state x_n^t iteratively for a particular time T, which results in approximately fixed-point value for $X(T) \approx X$ from the equation:3.1
2. Calculate the gradient of weights 'w' from loss in the equation:3.3
3. Update weights as per gradient found in the step 2.

After learning algorithm, the trained model can be used for supervised of semi-supervised tasks with implicit or hidden node features.

3.1.2 Limitations

Even if the vanilla GNN is efficient architecture for modelling graphical data structures, it also has some drawbacks that limit its performance.

1. The learning algorithm requires iteration for time T to update node states and compute the fixed-point value, which is a computationally expensive process.
2. The vanilla GNN uses the same parameter for each iteration, but other neural networks employ different settings in iteration, which enables them to mine hierarchical information from the neighbourhood.
3. It can't handle efficiently model graph structures when edges contain different feature information such as in knowledge graph message propagation depends on the kind of edges or relationship between nodes.

3.2 Graph Convolutional Network

The main aim of the graph convolutional network is to generalize the convolution process to be used for graph-based applications. Since CNNs are quite famous and successful in many applications of deep learning, so it is imperative to define convolution also for operations related to graphs. Methods are usually grouped into two as- spatial and spectral approaches.

Spectral methods are employed when the spectral representation of the graph is available. Here, graph specific Laplacian eigenbasis based filters are learned, so the model generated for a particular graph cannot be applied for other graph structures. In contrast, the spatial method tries to define convolution on spatially local nodes and the different size of neighborhood nodes.

As discussed in section 2.2, the geometric deep learning is the area that deals with interactions of the graph and neural networks. To understand typical graph convolutional networks, the first few basic graph theory should be known. Graphs are structured data with nodes and edges, where nodes and edges can have weights or labels as numerical or textual labels. Node features are a set of features that usually represent a node; understanding of labels should not be mixed with features. For an analogy- if the node represents a person, then the label can be a person's name, and features are a person's characteristics.

Some matrices are utilized to convert the information of the graph into some numerical format. These matrices are the Incidence matrix, Adjacency matrix (A), Degree matrix (D), and Laplacian matrix (or graph Laplacian; L=D-A). Laplacian matrix, also called Laplace Beltrami operator, measures smoothness, i.e., how quickly adjacency vertices change or how smooth the graph is.

Even if convolution is the backbone of CNN, wherein a 2D image grid/pixel values and neighborhood (8 pixels) are always fixed. But convolution fails on graphs because:

- Unlike images/grids, graphs do not have regular patterns for the neighborhood. Also, neighbor nodes can change their position, i.e., the locality principle is not valid.
- Graphs do not follow fixed node ordering, i.e., node number can be random at any time.

It is clear that a fixed convolution filter cannot be used here. Initially, the graph can be converted into adjacency matrix A and appended with feature vectors, as shown in Figure 3.2. But this method has issues as the Adjacency matrix will change if node ordering changes and the dimension of the matrix will change if graph size changes. So this form of graph can't be feed into neural networks as neural networks are designed for fixed dimensions.

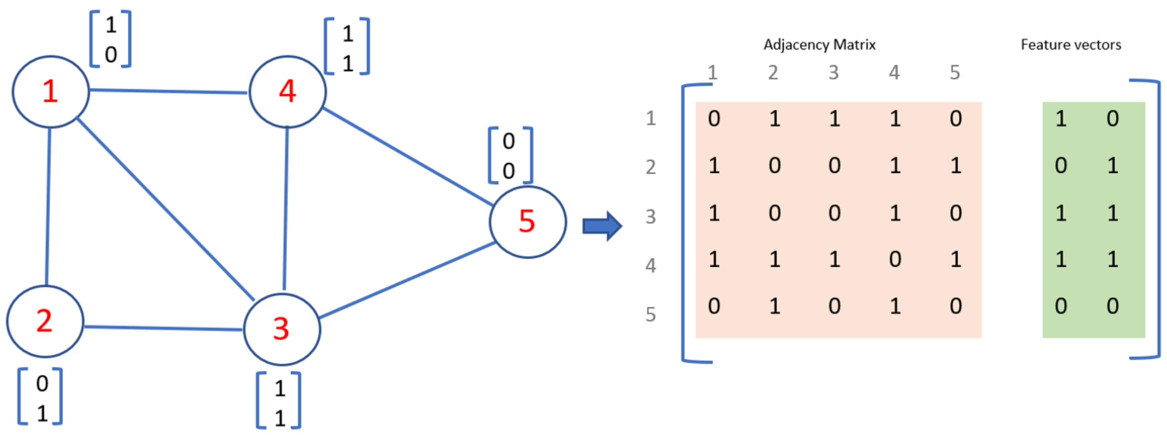


Figure 3.2: Representation of graph information as a combination of the adjacency matrix and feature vectors

(Kipf and Welling, 2016) suggest Graph convolutional networks, where convolution is for filter parameters that are shared for all nodes in the graph. This neural network is similar to CNN and employs all three steps (locality, aggregation, and composition) of basic CNN can be defined in the perspective of graphs, i.e.,

- **Locality**: how a node '1' is connected to its neighbor nodes '2', '3', '4'.
- **Aggregation**: how neighbor nodes '2', '3', '4' contribute to '1' with there own weight matrices, and how their weights are combined with corresponding nodes to contribute to the final node.
- **Stacking of layers**: how to perform a composition of functions, i.e., passing the result of aggregation to more complex layers.

In essence, most of the graph neural networks are similar in the overall architecture. The aim is to learn an embedding function of features or signals for a graph G=(V, E) (where V is vertices and E as edges) that accepts inputs as:

- x_n is a feature description for each node 'n', represented by feature matrix 'X' with dimensions NxM (N is the number the nodes, and M is the number of input features)

- ‘A’ is the adjacency matrix with size NxN, represents graph structure in a matrix

And, calculate result as a node-level output Z feature matrix with NxP dimension (P is the number of node-level output features). Also, the graph-level output can be generated by using some pooling operations. Here, the embedding function maps the nodes of graphs to a lower-dimensional embedding structure in such a way that similar nodes in graphs are embedded close to each other. This embedding function is also called encoder function ‘Y’ that projects graph to a lower-dimensional space, i.e., ‘Y’ operates on the actual graph and map this actual graph to d-dimensional space (d is much less than graph dimension) with retaining adjacent nodes in lower dimension feature space also. In Figure 3.3, in graph network nodes, u and v have embedded as Z_u and Z_v in d-dimensional embedding space, respectively. Here, similarity(dot product) between features of node u and v is the same as Z_a and Z_b. This embedding function ‘Y’ performs locality, aggregation, and stacking of layers altogether.

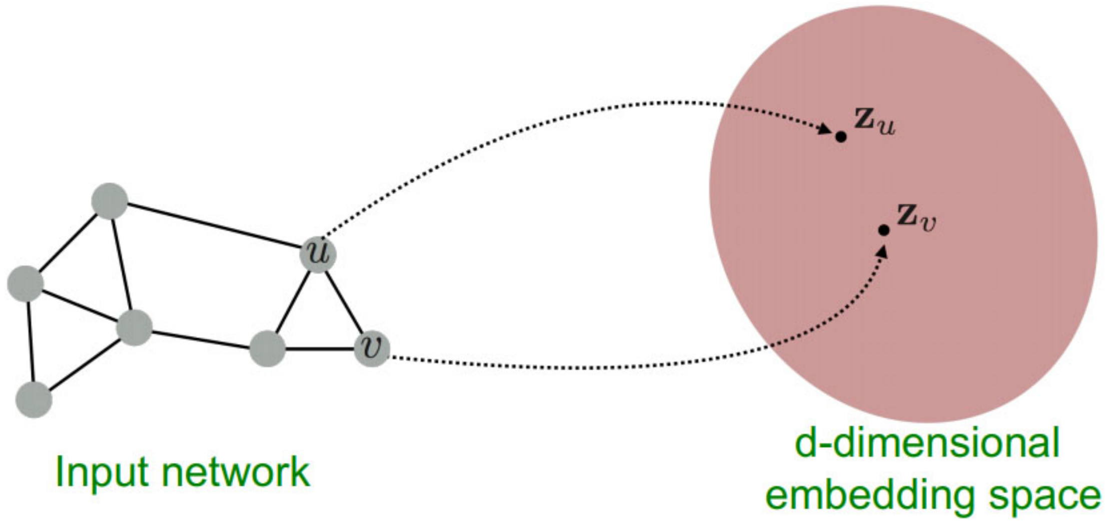


Figure 3.3: Embedding of nodes from a graph to lower dimensional space

The locality of information is achieved by forming a computational graph. In Figure 3.4, central node A with X_A feature is the target node, and its connections with the neighborhood of neighbors are determined using a computational graph. In a similar way, two layers computational graph of each node are generated.

Once computational graphs are available, the aggregation process starts to determine the contribution of Neighbour nodes, i.e., aggregation is performed using weights of there corresponding contribution. This is basically done by neural networks that are present

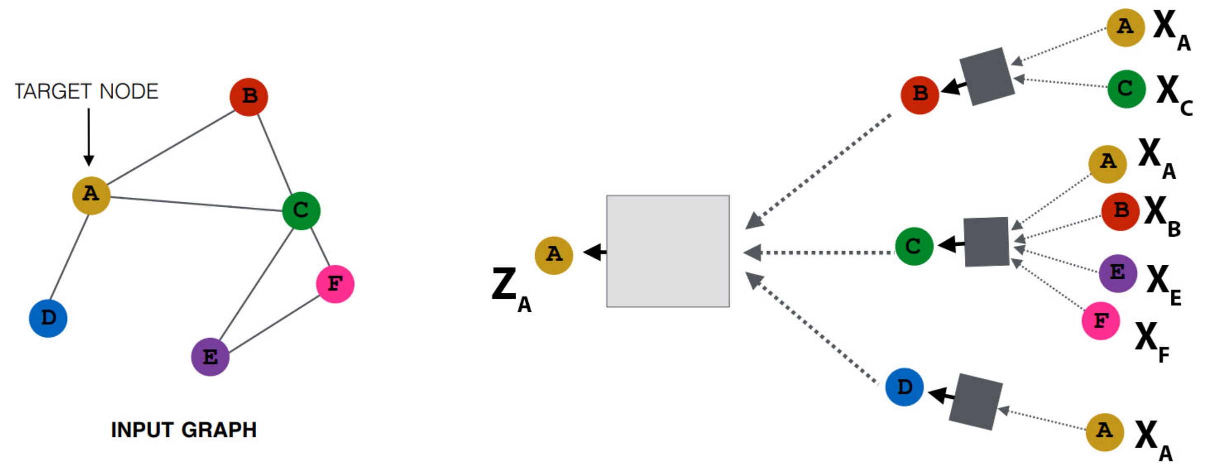


Figure 3.4: Computational graph for a target node

inside those blocks, as shown in Fig 3.4. For example, aggregation is done in blocks by summing X_A and X_C to form B and finally summing all of them to form A as output called Z_A using a bigger neural network. This summation operation is permutation invariant, i.e., it is irrelevant to node ordering. Thus neural networks have an important role in forming computational graphs.

Here, the forward propagation rule means information as X_a and X_b feature vectors from input side flows to output side Z_a with size Nx1, where N is much much less than M of neural network. Input nodes have their own embedding in feature space, which can be Mx1 for feature X_a.

For the mathematical formulation of propagation in GCN, every neural network can be represented by a non-linear function:

$$H^{l+1} = f(H^l, A) \tag{3.4}$$

Where embedding at layer l=0 is H^0 = X as initialization to node feature, and H^L = Z (where Z is node or graph level output, L is the number of layers). Any particular GCN model can be differentiated based on selection and parameterized form of function 'f'.

Considering a simple layer-wise propagation rule which is quite powerful although its simplicity is represented as:

$$f(H^l, A) = \sigma(AH^l W^l) \tag{3.5}$$

Where A is the adjacency matrix, W^l is a weight matrix for the l^{th} layer to be tuned in computational graphs and sigma as activation function (typically nonlinear in nature such as ReLU) that ensures final output belonging to each node is a vector of 'probabilities', where all values are in the range of 0 and 1, and the class of a node can be predicted as the element with maximum value.

This model has two fundamental limitations as for every node, feature vectors of neighborhood nodes are accumulated except that node itself, which can be fixed by adding a self-loop by simply enforcing an identity matrix. Also, adjacency matrix A is not normalized and multiplied with other vectors, which will, in turn, result in change dimensions of feature vectors and can cause an explosion in gradients and optimizer sensitive to feature scaling. This is fixed by normalizing Adjacency matrix A so that each row in A have some as 1. 'D' a diagonal degree matrix as D^{-1}A is added, which will result in averaging of neighboring node features. Now combining both fixes, a new propagation rule is formed as:

$$f(H^l, A) = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^lW^l) \tag{3.6}$$

where Â is A + I, where I is the identity matrix, and D̂ is the diagonal node matrix from Â. These generated embeddings are then supplied to a loss function, and a stochastic gradient is applied to train the weight parameters.

3.3 Graph Libraries and Frameworks

Several deep learning frameworks and libraries are available to model deep learning on graph and model graph neural networks. Apart from popular deep learning libraries such as TensorFlow, Keras, and PyTorch; this section introduces some tools that are available for implementing graph neural networks and are employed in this thesis work.

3.3.1 Deep Graph Library (DGL)

DGL is invented by distributed machine learning communities on GitHub; they are the same crew that released the XGBoost algorithm. It is a python package dedicated for implementing a family of graph neural networks. It supports seamless integration with traditional deep learning frameworks, i.e., if an end-to-end application involves the graph model component, then the rest of the logical ingredients can be written in any machine learning frameworks. As shown in Figure 3.5 (docs.dgl.ai, 2020), It is built atop existing layers of deep learning frameworks like, e.g., Pytorch, TensorFlow, or Apache MXNet, etc. This library provides a simplified set of GNNs specific functions to implement graph networks in a few lines of codes. DGL is actively maintained, supports cross-platform, contains readable code with proper documentation. It also provides efficient scalability for graphs with a large number of nodes or vertices.

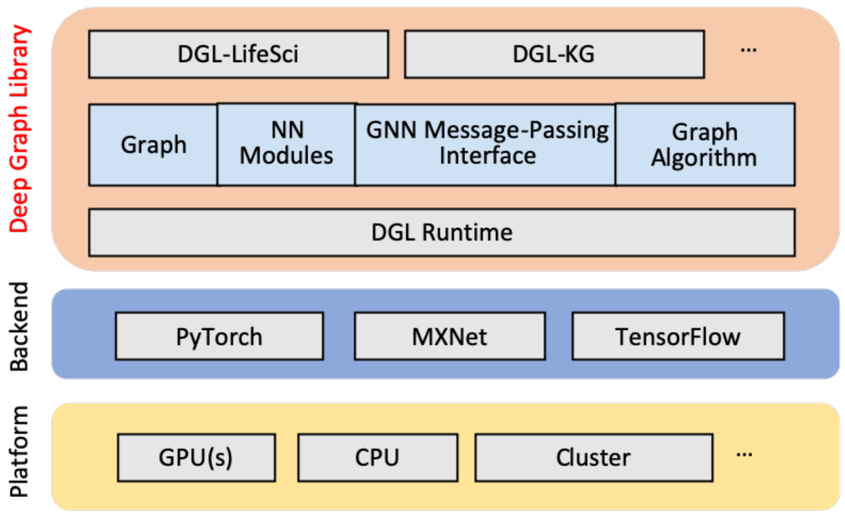


Figure 3.5: Deep Graph Library stack

AWS Deep Graph Network

DGL is available in Amazon web services based SageMaker service through AWS ECR(Elastic Container Registry) using a deep learning container where PyTorch and Apache MXNet are available as backend support. In Figure 3.6 (Sagemaker, 2020), layer-wise representation is shown for different modules available under AWS based in-depth graph network development.

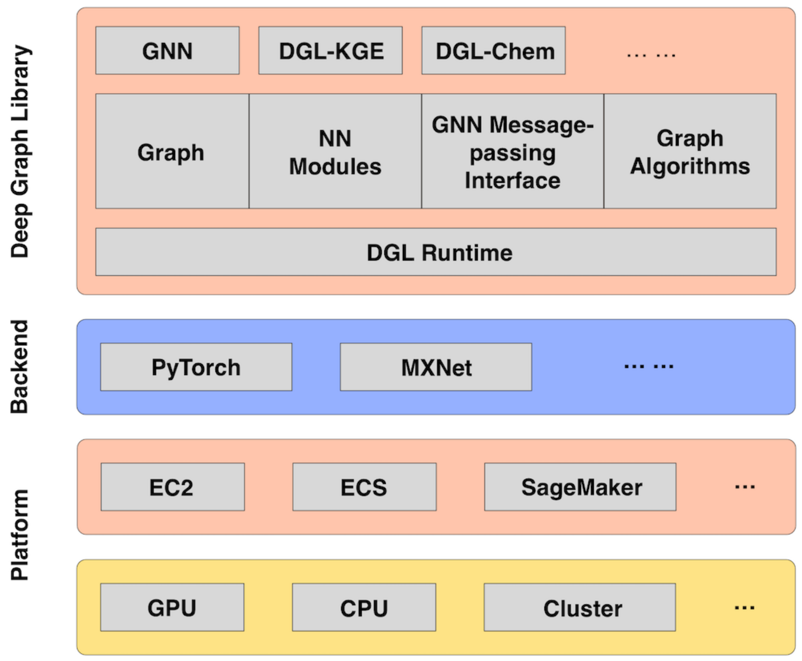


Figure 3.6: Deep Graph Library stack in AWS Sagemaker

3.3.2 Spektral

It is a framework designed for relational representation learning, which is built for python. This library also helps in modeling graph-based deep learning. It is based on TensorFlow 2 and Keras API. It provides a natural, fast, and flexible environment for implementing and experimenting graph neural networks(GNNs) (Grattarola and Alippi, 2020) by performing a broad set of functions. These functions allow loading of accessible datasets, processing of graphs as well as basic operations for GNNs, i.e., message-passing and pooling operations in convolution neural networks. It can be utilized to classify or clustering vertices or nodes in a graph network, creating new graphs with GANs and predicting edge or nodes properties. Being an open-source project on GitHub (Grattarola, 2020), it is currently the most mature library for implementing GNNs in the TensorFlow environment.

3.3.3 PyTorch Geometric (PyG)

It is a python library for deep geometric learning on irregular data such as graphs and manifolds. It is an extension to the PyTorch library in python. It has the latest types of graph networks already built and ready to call in a single line function. Apart from providing loader for several small and unique giant graphs, a variety of popular benchmark datasets and methods to transform graphs are available; it also contains many methods for relational representation learning and manifold data processing. In (Fey and Lenssen, 2019), the comparison is made with other deep graph libraries, and it is found that PyG is faster, even working with sparse data. It employs sparse GPU acceleration and dedicated CUDA kernels with efficiently handling input with different sizes to achieve a higher standard of data throughput.

3.3.4 StellarGraph

It is another python library for machine learning models on graph-structured data. It offers several state-of-the-art algorithms for machine learning models on graphs or networks. These algorithms can be used for a broad range of tasks such as node classification and interpretation, link prediction, classification of the whole graph. StellarGraph (StellarGraph, 2018) is based on TensorFlow2, high-level APIs of Keras, with NumPy and Pandas library. Also, it is designed to be integrated into existing workflows. Models implemented based on StellarGraph consists of a pair of items; such as, a data generator that converts basic graph structure and features from node to a standard input format which can be supplied to the Keras model for further processing, and the design of layers such as graph convolutional layer, dropout layer, and some dense layer.

3.3.5 Graph Net Library

It is a machine learning framework published by Deepmind research. Graph net library is built as a python library to allow the use of graphs networks (Battaglia, Hamrick, et al., 2018) in TensorFlow and Sonnet. Graph network framework utilizes the concept of the graph to graph modules where each graph contains characteristics as nodes, Relations among nodes, and system-level properties called Global attributes. It has a well-documented structure and has many google colab notebooks to showcase the usage of graph network libraries such as find the shortest path, sort numbers, and predict a physical system.

3.4 Overall Workflow

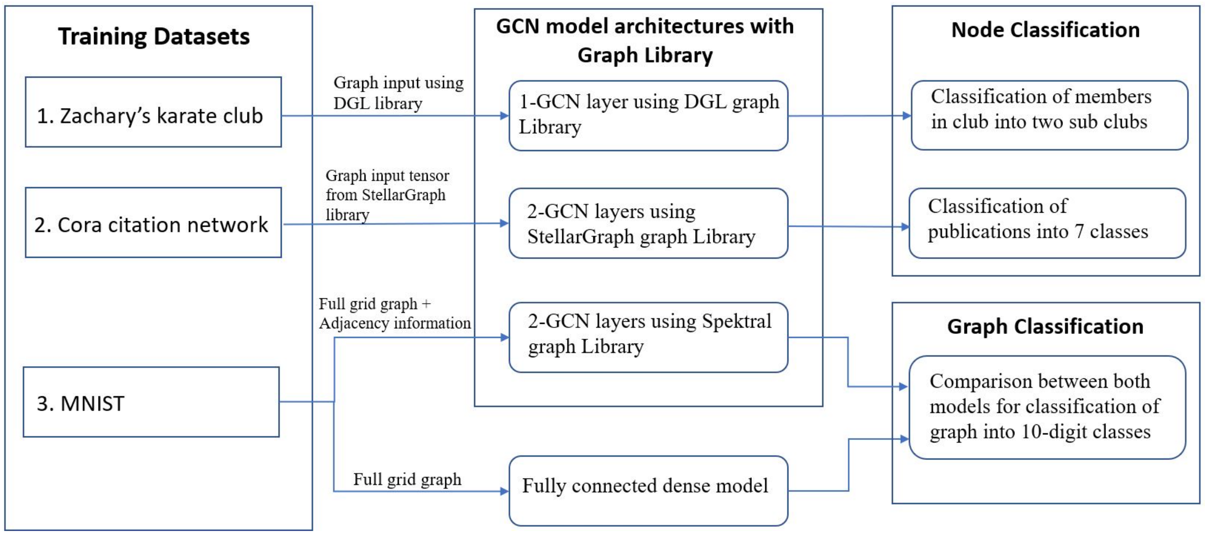

Training Datasets
1. Zachary's karate club
2. Cora citation network
3. MNIST
Graph input using DGL library
Graph input tensor from StellarGraph library
Full grid graph + Adjacency information
Full grid graph
GCN model architectures with Graph Library
1-GCN layer using DGL graph Library
2-GCN layers using StellarGraph graph Library
2-GCN layers using Spektral graph Library
Fully connected dense model
Node Classification
Classification of members in club into two sub clubs
Classification of publications into 7 classes
Graph Classification
Comparison between both models for classification of graph into 10-digit classes

Figure 3.7: High-level architecture for implementation design

Chapter 4

Implementation

Following the detailed design of graph neural network and mathematical formulation of propagation steps in chapter 3, this chapter implements those designs based on supported graph libraries for different application areas to conduct experiments aimed at research questions. Initially, technical setup, including the programming environment and required hardware, are covered, and details of three selected datasets used under this work are provided.

GCN neural network model is implemented on three chosen datasets, where two are for node classification, and one is for graph classification task. Jupiter notebooks are used for all three implementations; these notebooks are structured mainly in three sections:

1. Data preparation using standard python libraries such as Pandas and scikit-learn, which includes- generating graphical data from the dataset, performing some simple introspection, retrieving training, validation, and test data.
2. Design model by utilizing GCN layers from supported graph libraries such as DGL, StellarGraph, and Spektral.
3. Finally, the created model is trained, and the results are evaluated.

4.1 Overview

Since Graph convolutional networks are capable of solving a variety of problems, in this work node centric and graph centric applications are implemented.

4.2 Technical Setup

4.2.1 Programming Environment

As a programming environment setup, all implementation of modules is done in python3 language in Jupiter notebook format. Deep learning frameworks such as Tensorflow 2, Keras, NetworkX are used in combination with deep graph libraries such as DGL and Spektral. Operating systems are selected as windows, for local machine computation of Jupiter notebooks, Intellij Pycharm is used.

4.2.2 Hardware Requirement

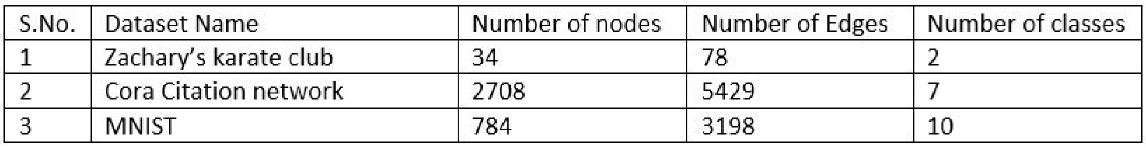
For computation-intensive implementation, NVIDIA GTX GPU is used locally to support with Intellij Pycharm IDE, and for DGL library-based computation, cloud AWS Sagemaker service is utilized, which provides- notebooks instances with high-end CPU capabilities optimized for machine learning tasks, platform-independent execution, and several Jupiter notebooks preloaded with kernels containing most of the deep learning frameworks.

However, in this thesis work, some basic graph datasets are used, but for larger and more complex graph structures, more powerful and customized hardware support is necessary.

4.3 Dataset

As Graph neural network with Convolution layer is studied in this work, for evaluation and testing GCN model, three benchmark datasets are considered. In this section, characteristics of datasets such as Zachary's karate club (club, 2013), Cora-citation network dataset (Sen et al., 2008), and MNIST are (LeCun and Cortes, 2010) are briefly described. Dataset statistics are summarized in Table 4.1.

Table 4.1: A summary of dataset statistics



S.No.	Dataset Name	Number of nodes	Number of Edges	Number of classes
1	Zachary's karate club	34	78	2
2	Cora Citation network	2708	5429	7
3	MNIST	784	3198	10

4.3.1 Zachary's Karate Club

It is a well known and documented dataset that represents a social network that includes 34 members and individual relationships outside the university karate club (club, 2013). It has a detailed depiction of community structure, which makes it quite popular. In this network of nodes, a densely connected set of nodes can be classified into groups. Specifically, the network can be organized into two communities, having centers as a karate teacher (node 0) and a club president (node 33), where network structure helps to predict how the karate club will be split into two. The club network is visualized in Figure 4.1, and color represents the community. This dataset is a small network of nodes and perfectly fits the scenario of structured data for graph application where data are naturally available in the graph structure.

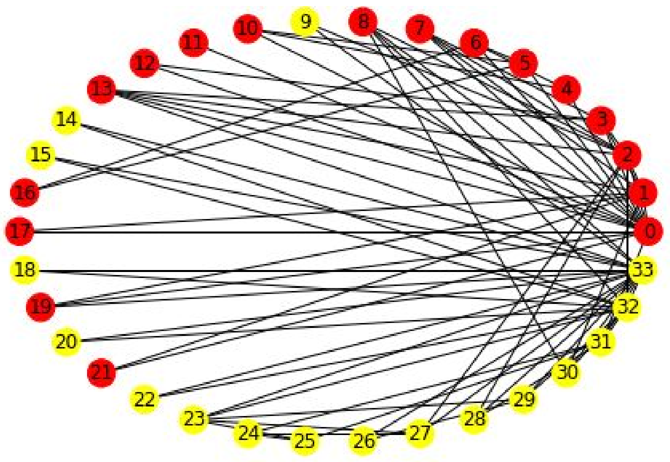


Figure 4.1: Graphical representation of Zachary's karate club dataset

4.3.2 Cora Citation Network

Among three citation network datasets - Cora, Citeseer, or Pubmed (Sen et al., 2008); the Cora dataset is selected as 2nd dataset in this work. This dataset is a collection of academic papers that represent the nodes and citation relationships between papers represents links/edge in graph network. Each node/paper is manually classified into seven classes/categories, such as Case-Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. The Cora Citation network depicted in the dataset has 2708 nodes as scientific publications in the machine learning domain, and 5429 edges/links; a high-level representation is shown in Figure 4.2 (Orbifold, 2020). The feature vector of each node is described by a 0/1 valued word vector that describes the presence/absence of corresponding words from the vocabulary that has a size of 1433, representing unique words regardless of frequency and order of appearance. Every publication has text and at least one node in the neighborhood with a moderate degree of citation as 2. This dataset is a large network of publications as nodes, i.e., data is naturally in a graph structure, so it can be considered in structured scenarios of graph application.

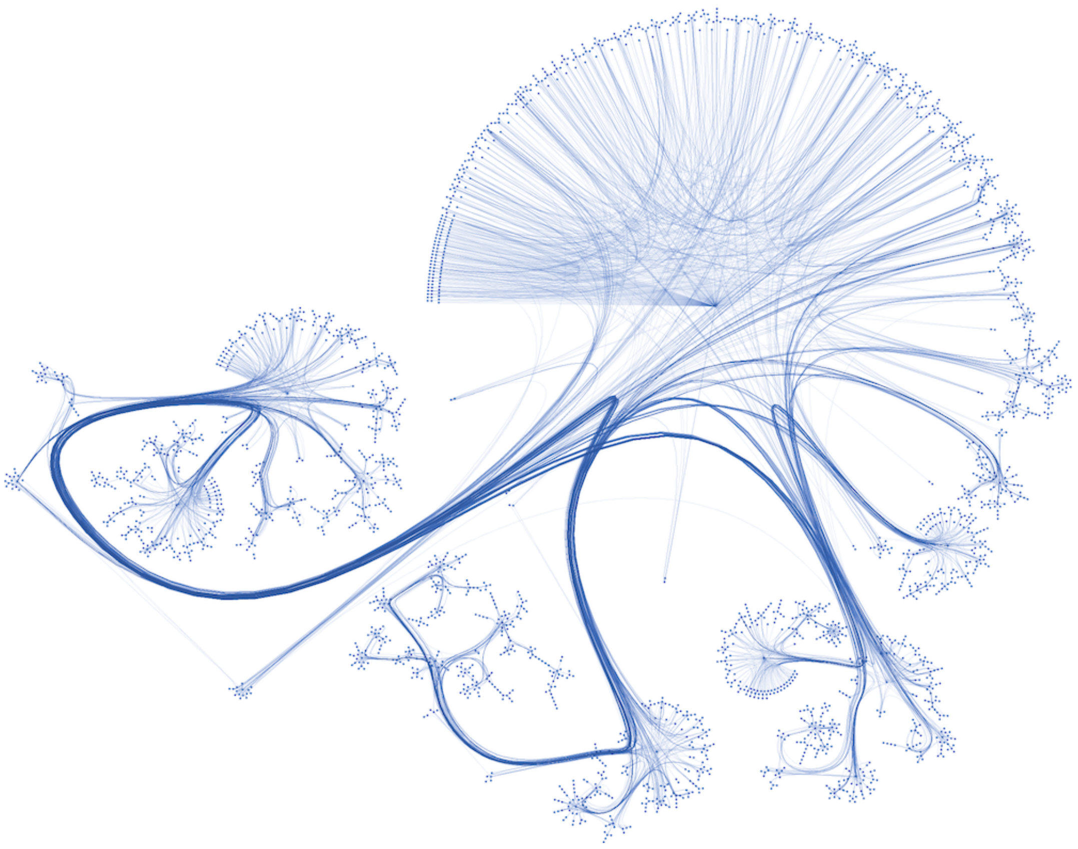


Figure 4.2: high-level representation of Cora citation network dataset

4.3.3 MNIST

It is one of the most popular and large datasets, which has the collection of handwritten digits with ten digits as classes. The MNIST dataset is actually a subset of the larger NIST dataset. The MNIST database (LeCun and Cortes, 2010) has training and test set with 60,000 and 10,000 examples images, respectively. All handwritten digits are normalized in size, stored within the fixed bounding box of image size (28x28 pixels), and anti-aliased that introduced the grayscale format. Due to its simplicity, the MNIST dataset is widely used for training and comparison of various machine learning techniques. The objective of those machine learning techniques is classifying handwritten digit into one of ten numeric characters.

Figure 4.3 shows a few sample images from the dataset and internal representations of one of these images. For matrix representation of images, If a pixel is black, its value is set as 1. Similarly, 0.1 and 0 value represents grey and white pixels, respectively. It can also

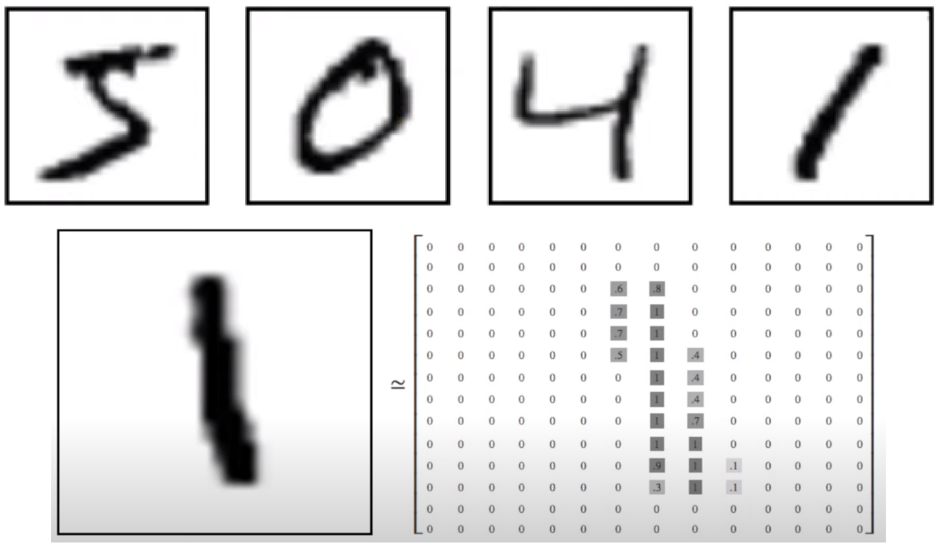


Figure 4.3: MNIST sample images and matrix-based internal representation

be converted into the vector by flattening row/column-wise, giving vector size as 784x1, i.e., each image/data point is a column vector of shape 784x1. MNIST contains images of fixed size, so it can be considered in non-structured scenarios for graph applications, where data are not in graph structure naturally.

4.4 Node Classification in Zachary's Karate Club and Cora Dataset

Experiment 1: Zachary's Karate Club Dataset

In the first experiment, graph convolutional network is used to predict the classification of nodes/members in karate club into two by utilizing the network structure,i.e., predicting the tendency of each member in the club to join a side (0 or 33) utilizing an available social network of all members in the club. For a simple propagation, A and X represents the adjacency matrix and feature vector for nodes, respectively. Different graph problems discussed under DGL library (docs.dgl.ai, 2020) are referred for implementing this method of node classification in a graph using GCN layers.

1. On the local machine, using Jupiterlab from Anaconda Navigator, a python three Jupiter notebook is used to implement a graph convolutional model for the karate club dataset.

2. Initially, the Karate club dataset can be imported from a python package called NetworkX or created as two arrays as the source and destination endpoints. The second method is used here to build a karate club graph.

3. Since the input feature vector is not available with the dataset for training, adding features to nodes and edges are added using a learnable embedding vector from the 'nn' module of PyTorch package, making 34 nodes having embedding dimensions of 10.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

new_embed =nn.Embedding(34, 10) # adding 10 features to each of the 34
                                                  nodes
club_graph.ndata['feat'] =new_embed.weight
# Input features of node 0, 17 and 33
print(club_graph.ndata['feat'][[0, 17, 33]])
```

4. Graph convolutional networks are defined as using the' GraphConv' module from the DGL.nn module. Information of each node with neighborhood features is encoded so that each node is represented as aggregate if neighborhood nodes. Also, a self-loop is added ($\hat{A} = A + I$) for each node for including its own features, and degree matrix 'D' is added to normalize '$\hat{A}$' to make encoded function as $f(X,A) = D^1 AX$.

5. GCN layers are stacked to form the output of dimensions 34x2. Here, Input features with size ten are transformed by the first layer to a hidden size of 7, and the second layer generates output features of size 2 representing two sub-clubs.

6. Labels for node 0 and node 33 are added because, in the dataset, instructor(node 0) and president(node 33) of the club are already decided.

7. Then, the created network model is trained similar to training a PyTorch model in the loop (epochs), i.e., the model optimizer is created first, inputs feature supplied

to model, and then based on loss calculated model is automatically upgraded with weights. After the loop, the optimized model is generated, and intermediate states are visualized.

Experiment 2: Cora-Citation Network Dataset

In this experiment, the semi-supervised classification of publications/papers in the citation network is performed using a graph convolutional network. Here, a GCN model is trained to predict the label or class of a node in the citation network among seven available classes, which is a type of node classification in a graph network. Following the transduction learning method, for training, all nodes and edges are observed with few labels, and the target is to find missing labels. Also, the resulting model is used to represent vector embedding for each text document. This implementation uses Graph Convolution Network (Kipf and Welling, 2016) method, StellarGraph GNN library is used, and implementation is followed as per steps provided in (StellarGraph, 2018) documentation and examples.

1. A Notebook instance is created on AWS Sagemaker for using Jupiter notebook, where several deep learning kernels are available. Here, 'conda_tensorflow_p36' kernel is used that provides preloaded TensorFlow, Keras, and other libraries.
2. StellarGraph library is installed using the standard python package management system.
3. After importing the required libraries, the Cora dataset is loaded from the StellarGraph object using dataset loader 'cora()'. Graph and ground-truth classes for nodes are retrieved.
4. 4. Dataset is split into the two subsets of nodes, one for training and another for the validation and testing purpose. Here, 300 nodes are used for training, 500 nodes are used for the validation process, and the rest nodes are kept for testing purposes.
5. Categorical target data is converted to the numerical arrays using a one-hot encoder, and it will be compared to the soft-max output of the model generated. The method

called ‘LabelBinarizer()’ from scikit-learn is employed to perform this conversion.

6. 6. Now graph convolution layer is created as per StellarGraph architecture, i.e., create a data generator for converting graph structure into matrix format that can be supplied as input to Keras model and design layers. ‘FullBatchNodeGenerator’ class with ‘gcn’ method from StellarGraph is used as a data generator. For layer design, ‘GCN’ class is used for 2 hidden GCN layers, and activation is applied on the result of each GCN layer. Here, the dropout rate as 0.5, and two hidden layers with size 64 and 32 are used. Input and output tensors of the GCN model are exposed to fit into the Keras model. An Additional dense layer is added as required for Keras model functionality to support the output of a 32-dimensional vector.

```
generator =FullBatchNodeGenerator(G, method="gcn")
train_gen =generator.flow(train_subjects.index, train_targets)
gcn =GCN(
layer_sizes=[64, 32], activations=["relu", "relu"], generator=generator,
                                             dropout=0.5)
x_inp, x_out =gcn.in_out_tensors()
```

7. The Keras training model is generated with the learning rate as 0.01, the input and output tensors from the last step, where output tensor acts as the predictions for the dense layer. Early Stopping functionality is also added to stop the training process if improvement in validation accuracy does not happen anymore. Finally, the model is trained after setting layers, training data, validation data, and some training callbacks.

8. The accuracy metrics and the behavior loss function are visualized using the ‘plot_history()’ method of StellarGraph. The generated model is evaluated on the test data using ‘evaluate()’ function.

9. Prediction is done for all nodes using the trained model and observed for class value for each node.

10. Finally, the embedding model is generated from the prediction model. Since the output dimension of the last GCN layer is 32, so to plot this data dimension is reduced to 2 using the dimensionality reduction method ‘TNSE’ from scikit-learn,

and vector embedding for every node/text document is visualized on 2-dimensional space.

4.5 Graph Classification in MNIST Dataset

Experiment: MNIST Dataset

In this experiment, graph signal classification is performed on MNIST dataset. In (Defferrard, Bresson, and Vandergheynst, 2016) generalizes CNNs from low-dimensional and fixed grid structure to high-dimensional and irregular structures such as networks or graphs by creating a grid graph of connected neighbors pixels. This 2-dimensional grid graph can also be used in the GCN model to depict spatial information of each image in the dataset. This implementation compares the baseline performance of GCN architecture by (Kipf and Welling, 2016) with a fully connected model without graph convolutional, where both methods are utilizing a 2-dimensional grid suggested in (Defferrard, Bresson, and Vandergheynst, 2016). Here, the Spektral library is employed to design the graph convolution network for classifying the input digits based on the intensity values representing the nodes and the neighborhood relationships considered as the edges.

1. Jupiter notebook is created on AWS Sagemaker, required libraries are imported, and Spektral library is installed using the standard python package management system.

2. From 'spektral.dataset' module, MNIST dataset is imported using 'load_data()' method and split into training, validation, and testing data with labels.

3. The adjacency matrix is retrieved from the dataset, and visualization is generated as a 2-dimensional array. Also, the grid graph is generated as per (Defferrard, Bresson, and Vandergheynst, 2016) and visualized using adjacency information.

4. A Meshgrid for a particular number is displayed to understand the topology and actual graph structure with numbered nodes and edges between them.

5. In the model building, first, the densely connected Neural network is designed. The Flattened input array of 784 nodes are supplied to a regular densely-connected Neural layer, which is designed using ‘dense()’ method of ‘keras.layers’ module. The output channel of the dense layer is set as 10, representing labels of the dataset.

6. Then, designed a fully connected neural network model is trained and evaluated on test data.

7. Similarly, the second model is designed for Graph convolutional network for classification with different feature graphs. Initially, the normalized Laplacian is computed using the ‘preprocess()’ method from ‘spectral.GraphConv’ class, which represents the graph convolution filter. Here, the input channel size is 784, and two graph convolutional layers with 32 output channels are added using ‘GraphConv’ class from the Spektral library. Finally, the dense layer is added with 10 output channels as digit classes.

```
X_in =Input(shape=(N, F))
A_in =Input(tensor=sp_matrix_to_sp_tensor(fltr))

graph_conv_1 =GraphConv(32, activation='elu',
                     kernel_regularizer=l2(l2_reg))([X_in, A_in])
graph_conv_2 =GraphConv(32, activation='elu',
                     kernel_regularizer=l2(l2_reg))([graph_conv_1, A_in])
flatten =Flatten()(graph_conv_2)
fc =Dense(512, activation='relu')(flatten)
output =Dense(n_out, activation='softmax')(fc)

# Build model
model =Model(inputs=[X_in, A_in], outputs=output)
optimizer =Adam(lr=learning_rate)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
                                          metrics=['acc'])
model.summary()
```

8. Then, the graph convolutional network is trained and evaluated. The test scores of both models are captures and compared.

9. The trained weights and the intermediate output of the GCN model are visualized.

Chapter 5

Results and Analysis

5.1 Overview

In this chapter, the intermediate outputs, final results, their analysis are presented as part of experiments on all three datasets using graph-based deep learning methodology. The classification of nodes and graphs using different graph libraries are evaluated based on the performance metrics. It also includes critical analysis based on resultant classification metrics and some comparison as applicable, which depicts insights about addressed research questions in this work.

5.2 Analysis of Classification Methods and Results

5.2.1 Experiment on Zachary's Karate Club Dataset using DGL Library

The Zachary's karate club network is a structured graph scenario. The initial graph structure is represented using in Figure 5.1, where each node with the same colour depicts individual members in the club, and these members are supposed to be joining any of sub-club led by the instructor(node 0) and the president (node 33).

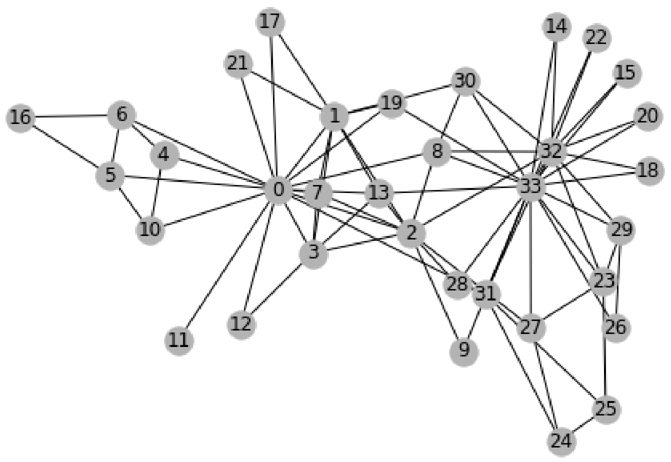
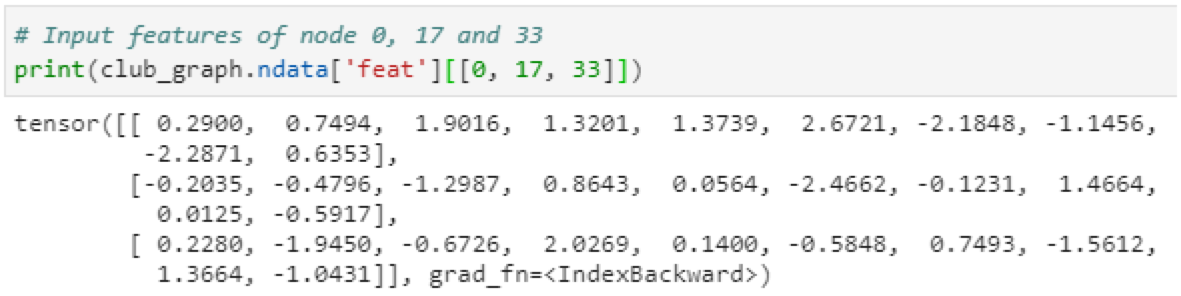


Figure 5.1: Initial Graph structure of Zachary's karate club network dataset

Figure 5.2 shows feature tensor for nodes as learnable embedding vectors, where nodes are embedded with 10 features each. These features represent the several relationship values of nodes/members with the instructor and the president in the club graph, and based on these values, node classification is predicted.

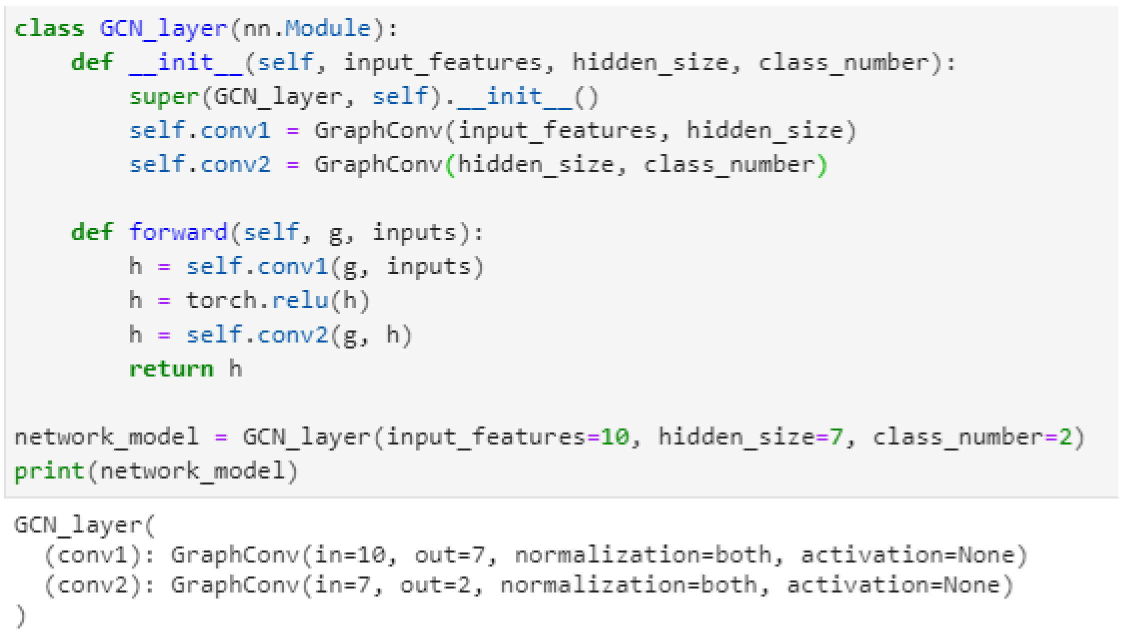


```
# Input features of node 0, 17 and 33
print(club_graph.ndata['feat'][[0, 17, 33]])
```

```
tensor([[ 0.2900,  0.7494,  1.9016,  1.3201,  1.3739,  2.6721, -2.1848, -1.1456,
         -2.2871,  0.6353],
        [-0.2035, -0.4796, -1.2987,  0.8643,  0.0564, -2.4662, -0.1231,  1.4664,
          0.0125, -0.5917],
        [ 0.2280, -1.9450, -0.6726,  2.0269,  0.1400, -0.5848,  0.7493, -1.5612,
          1.3664, -1.0431]], grad_fn=<IndexBackward>)
```

Figure 5.2: Feature tensor assigned to each node in karate club network

The created GCN network model is presented here in Figure 5.3, where two graph convolutional layers are utilized to make the decision between two classes. Then, this model is trained as per the Keras model using optimizer and for 100 epochs. With subsequent epochs, computation loss gradually decreases and nodes are classified more prominently; these epochs and loss value are added in Appendix A.



```
class GCN_layer(nn.Module):
    def __init__(self, input_features, hidden_size, class_number):
        super(GCN_layer, self).__init__()
        self.conv1 = GraphConv(input_features, hidden_size)
        self.conv2 = GraphConv(hidden_size, class_number)

    def forward(self, g, inputs):
        h = self.conv1(g, inputs)
        h = torch.relu(h)
        h = self.conv2(g, h)
        return h

network_model = GCN_layer(input_features=10, hidden_size=7, class_number=2)
print(network_model)
```

```
GCN_layer(
  (conv1): GraphConv(in=10, out=7, normalization=both, activation=None)
  (conv2): GraphConv(in=7, out=2, normalization=both, activation=None)
)
```

Figure 5.3: GCN network model summary for Zachary's karate club dataset

The intermediate Keras logits are saved for visualization of graph state in each epoch. In Figure 5.4, a few intermediate states are displayed from Epoch 0 to 99. The GCN model started with a few nodes which are classified into two groups in epoch 0 and colored differently. Gradually, node 0 and node 33 is becoming two distant centers in the graph structure. It can also be interpreted that members/nodes are classified, and their distance is also changing to a certain limit with each epoch; these changes in distance represent closeness in relationship to the instructor (node 0) and the president (node 33). This shows GCN layers based on DGL are efficient in node classification for this small club network.

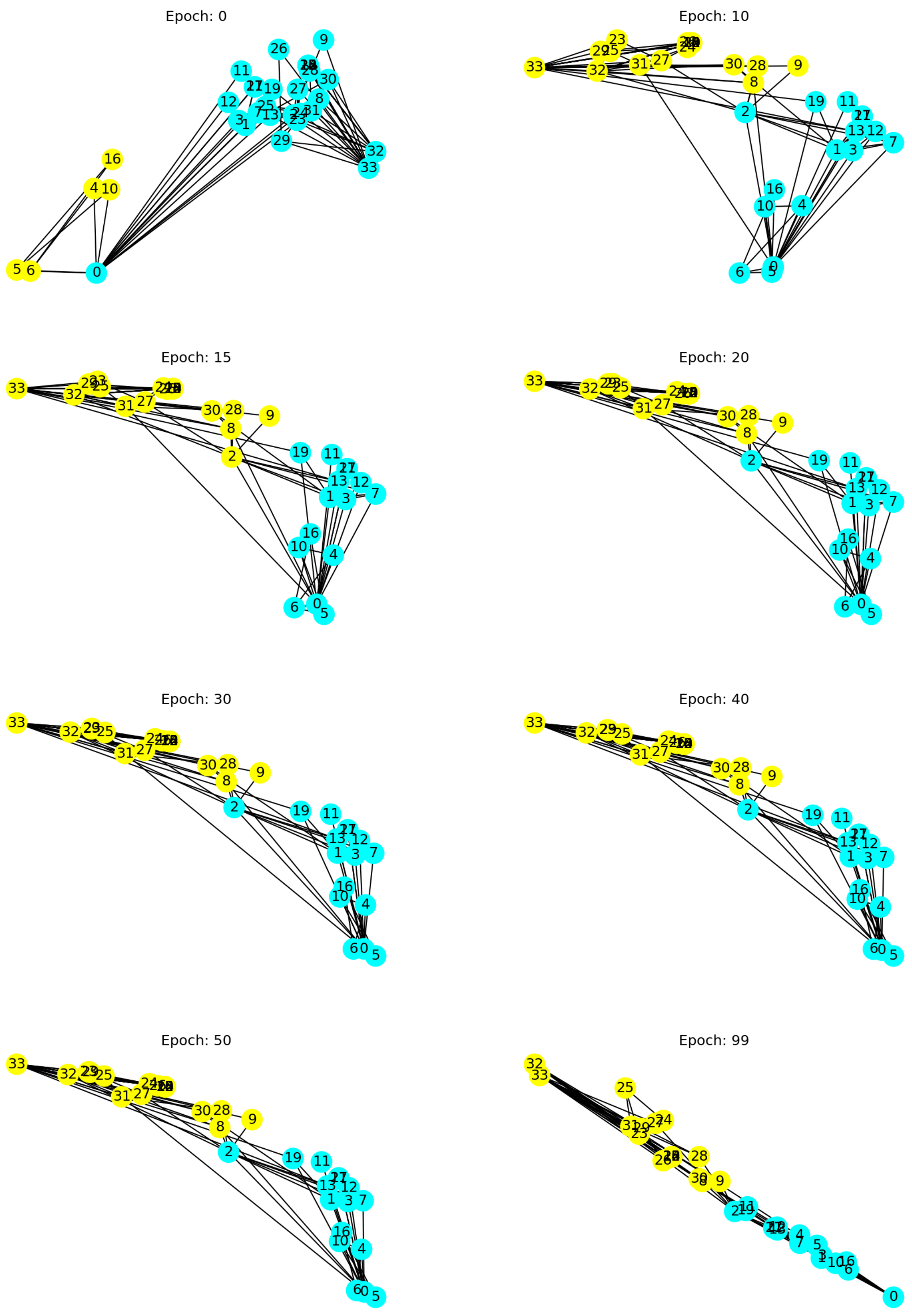
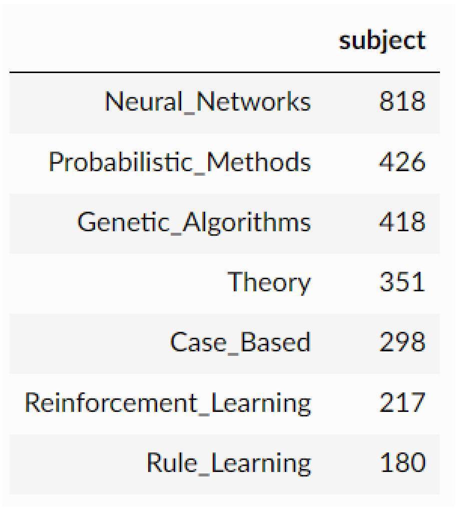


Figure 5.4: Intermediate states of the graph in the karate club network while training GCN model

5.2.2 Experiment on Cora Citation Network Dataset Using StellarGraph Library

Cora citation network dataset is also a structured graph scenario and contains 7 classes of academic publications. In this experiment, the performance of GCN layers supported by the StellarGraph library is tested. Cora dataset is imported, graph information, and the number of text documents under each category is shown using the StellarGraph library in Table 5.1. Here, the graph is considered as undirected one, and the edge is referencing a paper that is being cited.

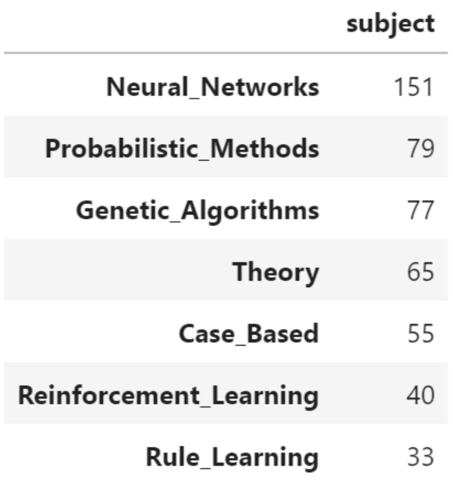


	subject
Neural_Networks	818
Probabilistic_Methods	426
Genetic_Algorithms	418
Theory	351
Case_Based	298
Reinforcement_Learning	217
Rule_Learning	180

Table 5.1: Document classes in Cora citation network dataset

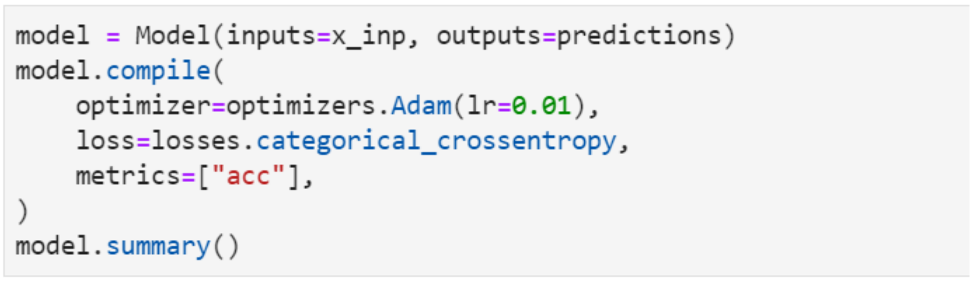
The dataset is split into training, validation, and test group. The training set size is 500, and Table 5.2 shows the count for documents in each class.

Table 5.2: Document classes and count in training set for Cora citation network dataset



	subject
Neural_Networks	151
Probabilistic_Methods	79
Genetic_Algorithms	77
Theory	65
Case_Based	55
Reinforcement_Learning	40
Rule_Learning	33

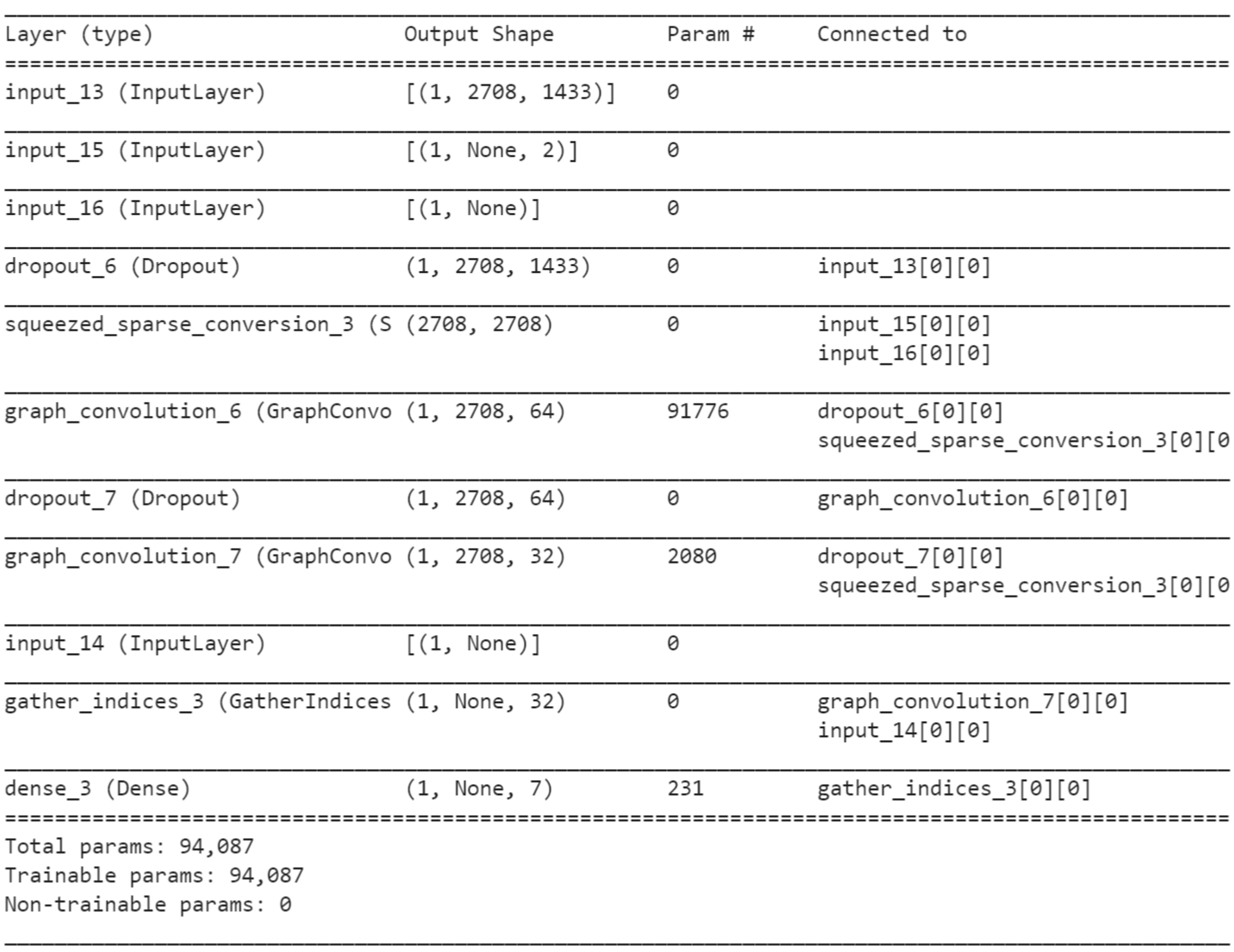
Training Model is designed based on the GCN layer provided in the StellarGraph library, with two-layer as 64 and 32 output channels. The last dense layer is responsible for generating the classification of documents in 7 classes. The designed GCN model and summary are presented in Figure 5.5 and Table 5.3.



```
model = Model(inputs=x_inp, outputs=predictions)
model.compile(
    optimizer=optimizers.Adam(lr=0.01),
    loss=losses.categorical_crossentropy,
    metrics=["acc"],
)
model.summary()
```

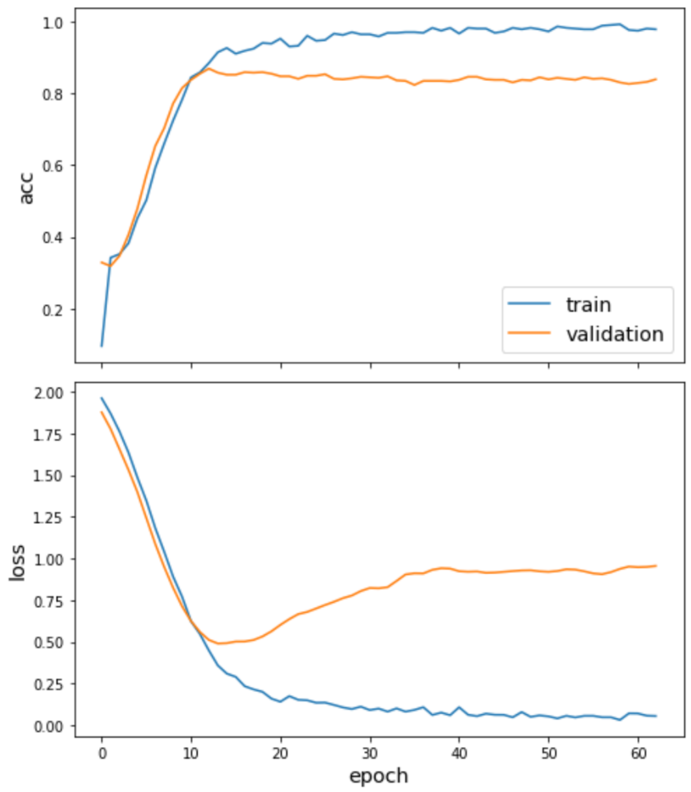
Figure 5.5: GCN network model design Cora citation network dataset

Table 5.3: GCN network model summary for Cora citation network dataset



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_13 (InputLayer)           [(1, 2708, 1433)]    0
__________________________________________________________________________________________________
input_15 (InputLayer)           [(1, None, 2)]       0
__________________________________________________________________________________________________
input_16 (InputLayer)           [(1, None)]          0
__________________________________________________________________________________________________
dropout_6 (Dropout)             (1, 2708, 1433)      0           input_13[0][0]
__________________________________________________________________________________________________
squeezed_sparse_conversion_3 (S (2708, 2708)         0           input_15[0][0]
                                                                 input_16[0][0]
__________________________________________________________________________________________________
graph_convolution_6 (GraphConvo (1, 2708, 64)        91776       dropout_6[0][0]
                                                                 squeezed_sparse_conversion_3[0][0
__________________________________________________________________________________________________
dropout_7 (Dropout)             (1, 2708, 64)        0           graph_convolution_6[0][0]
__________________________________________________________________________________________________
graph_convolution_7 (GraphConvo (1, 2708, 32)        2080        dropout_7[0][0]
                                                                 squeezed_sparse_conversion_3[0][0
__________________________________________________________________________________________________
input_14 (InputLayer)           [(1, None)]          0
__________________________________________________________________________________________________
gather_indices_3 (GatherIndices (1, None, 32)        0           graph_convolution_7[0][0]
                                                                 input_14[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (1, None, 7)         231         gather_indices_3[0][0]
==================================================================================================
Total params: 94,087
Trainable params: 94,087
Non-trainable params: 0
__________________________________________________________________________________________________
```

After training the designed Keras model with selected training and validation set, the performance metrics such as accuracy and loss are determined, and intermediate historical values for accuracy and loss is plotted w.r.t number of epochs are shown in Figure 5.6. From the 'accuracy vs. epochs' plot, it is clear that model accuracy reached around maximum in the first 20 epochs, and further training does not improve this result by a large difference. Also, from the 'loss vs. epochs' plot, it can be understood that performance is comparable for both training and validation data for early 15 epochs, but plots start departing consistently to a limit and stop increasing after that, this indicates that training should be stopped earlier. This shows the importance of retrieving metrics during the training of the model.



```
Test Set Metrics:
        loss: 0.4656
        acc: 0.8654
```

Figure 5.6: Intermediate accuracy and loss vs. epochs plot with test metric results of the GCN model for Cora citation network dataset

This trained prediction model is then utilized to make predictions on remaining test data, and the top 10 results are shown in Table 5.4.

Table 5.4: GCN network model predictions for Cora citation network dataset

	Predicted	True
31336	Neural_Networks	Neural_Networks
1061127	Rule_Learning	Rule_Learning
1106406	Reinforcement_Learning	Reinforcement_Learning
13195	Reinforcement_Learning	Reinforcement_Learning
37879	Probabilistic_Methods	Probabilistic_Methods
1126012	Probabilistic_Methods	Probabilistic_Methods
1107140	Reinforcement_Learning	Theory
1102850	Neural_Networks	Neural_Networks
31349	Neural_Networks	Neural_Networks
1106418	Theory	Theory

For generating node embedding of each node from the dataset, 32 output channels of designed GCN layers are reduced to 2 to fit into 2-dimensional space, as shown in Figure 5.7. Here each color represents an academic publication class. It can be seen that few classes are more clustered, and some are compactly classified into groups. With this clear node classification, it is evident this GCN layers can also be utilized for such datasets with implicit features.

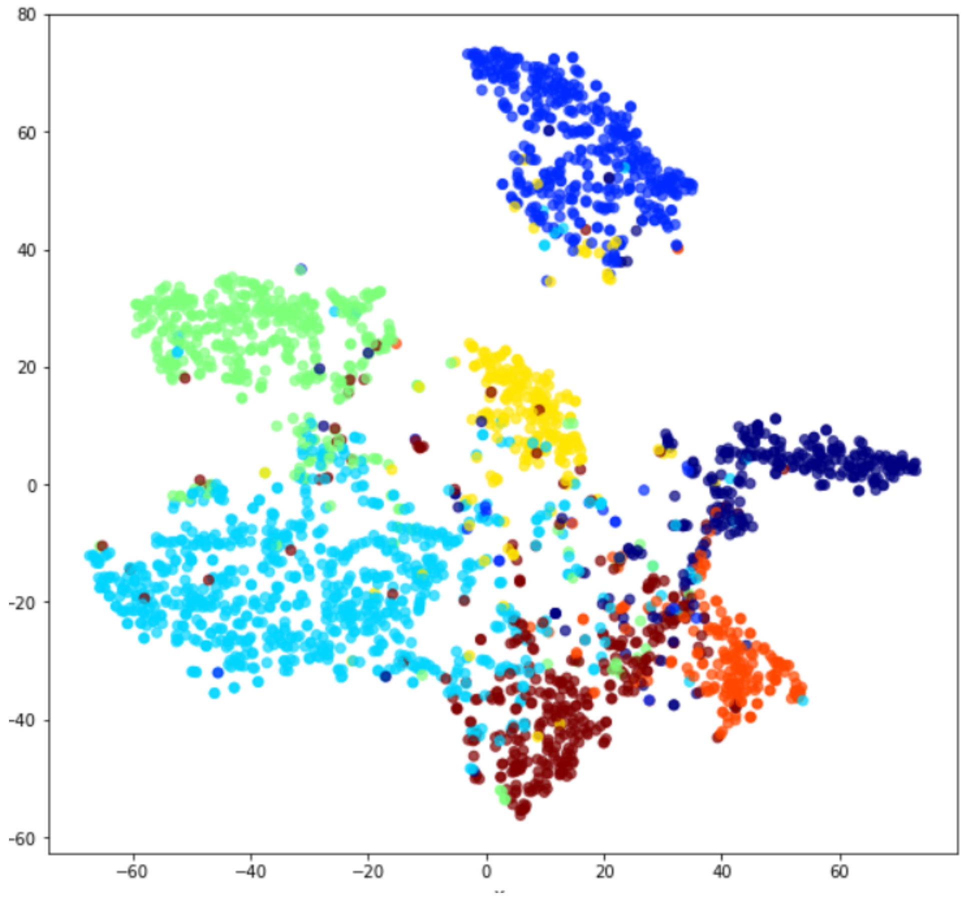
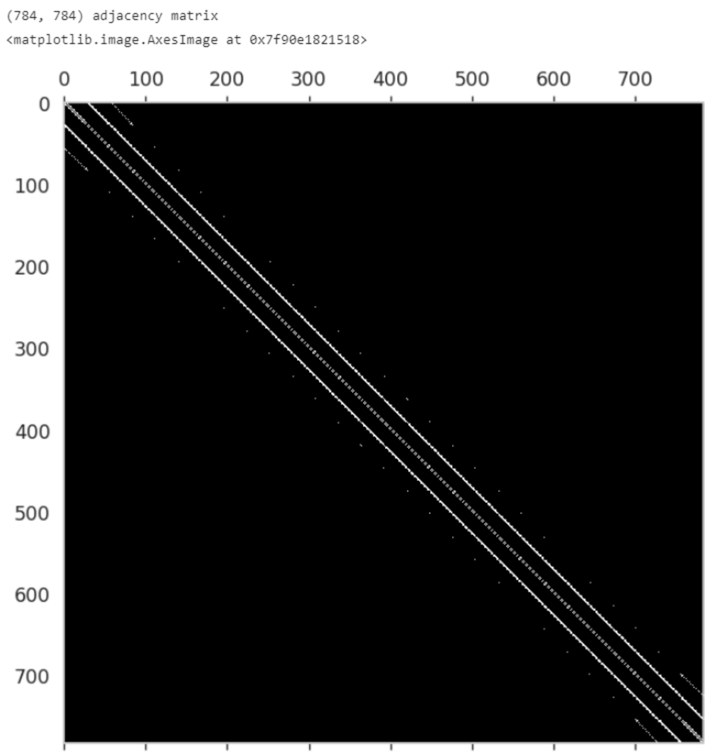


Figure 5.7: Node embedding visualization for each publication of Cora citation network dataset

5.2.3 Evaluation and Comparison of Experiment on MNIST Dataset

The MNIST dataset is a collection of images that can be considered as a non-structured scenario because the dataset is not in graph form naturally. But, using graph library, this dataset is processed to make predictions on images into 10 digit classes. Since a GCN layer accepts input in the non-Euclidean domain, images data are supplied to the model with adjacency information and grid graph structures. In this experiment, the adaptability and performance of GCN layers supported by the Spektral library are tested. Here, the aim is to classify the digits images using the intensity pixel values as nodes and relationships with Neighbour nodes as edges.

The list of adjacency matrices returned by the 'mnist.loaddata()' method is visualized in Figure 5.8.



```
(784, 784) adjacency matrix
<matplotlib.image.AxesImage at 0x7f90e1821518>
```

Figure 5.8: 2-D plot for adjacency matrix for MNIST dataset

Using this adjacency list, the grid graph is generated with 784 nodes and 3198 edges. As shown in Figure 5.9, it can be seen that in the four corners of the generated grid graph has some artifacts.

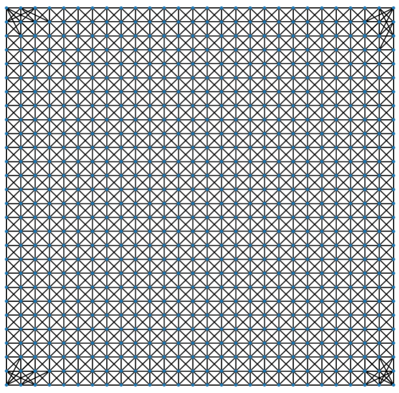
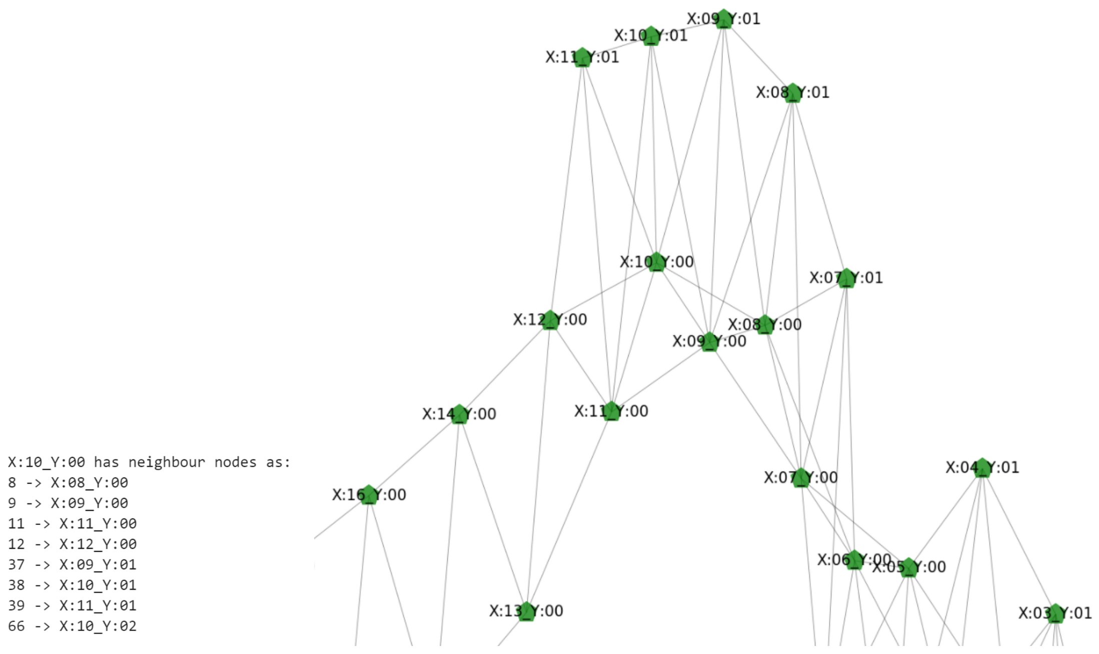


Figure 5.9: Full grid graph generated for MNIST dataset

To understand the topology of the graph with details, some labeled nodes and connections are presented in Figure 5.10, where the eight neighbors of node ‘X:10_Y_:00’ are shown. An overall topological structure, as shown in Figure 5.11, which is created as a new graph from retrieved adjacency matrics using Scipy sparse matrix.



```
X:10_Y:00 has neighbour nodes as:
8 -> X:08_Y:00
9 -> X:09_Y:00
11 -> X:11_Y:00
12 -> X:12_Y:00
37 -> X:09_Y:01
38 -> X:10_Y:01
39 -> X:11_Y:01
66 -> X:10_Y:02
```

Figure 5.10: Neighbours in Full grid graph generated for MNIST dataset

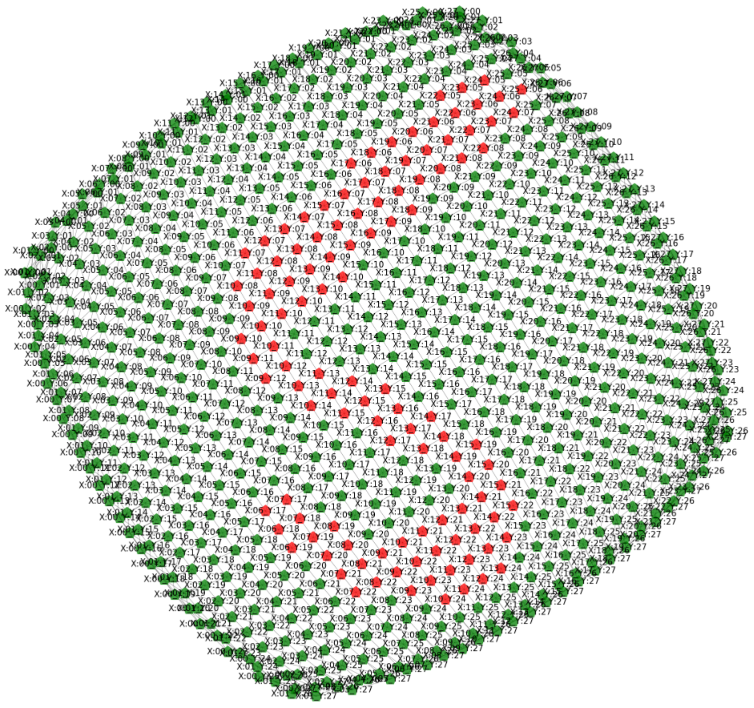
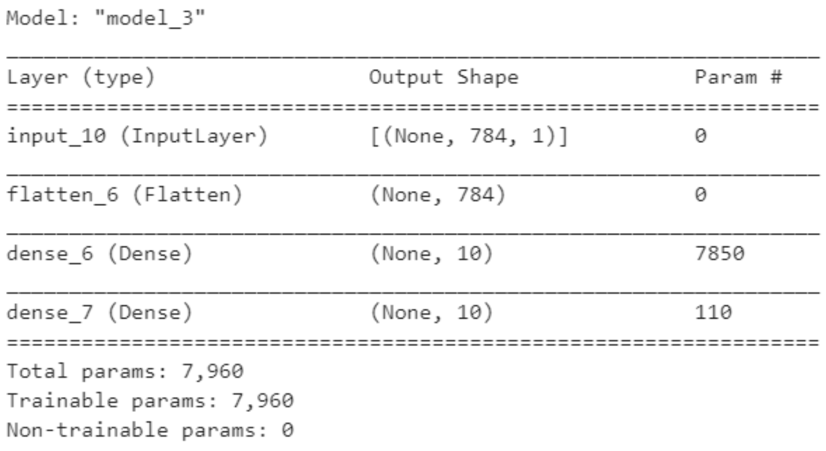


Figure 5.11: Overall topological structure of a graph data for MNIST dataset

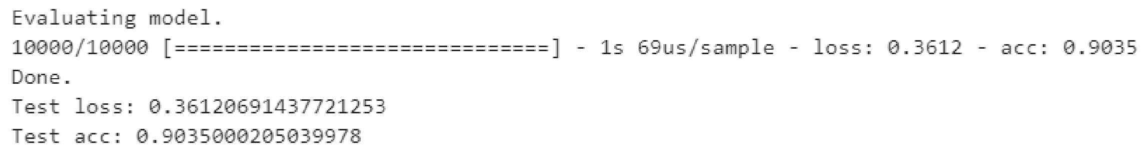
As the first part of the experiment, a fully connected dense model is created using 10 output channels, and summary is shown in Table 5.5, where 10 output channel is responsible for classifying input into 10 digits.

Table 5.5: Summary of the fully connected dense model for MNIST dataset



```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(None, 784, 1)]          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                7850
_________________________________________________________________
dense_7 (Dense)              (None, 10)                110
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
```

After the training model with the training and validation set, the model is tested on 10000 test data, and the result is shown in Figure 5.12; total loss and accuracy are measured.

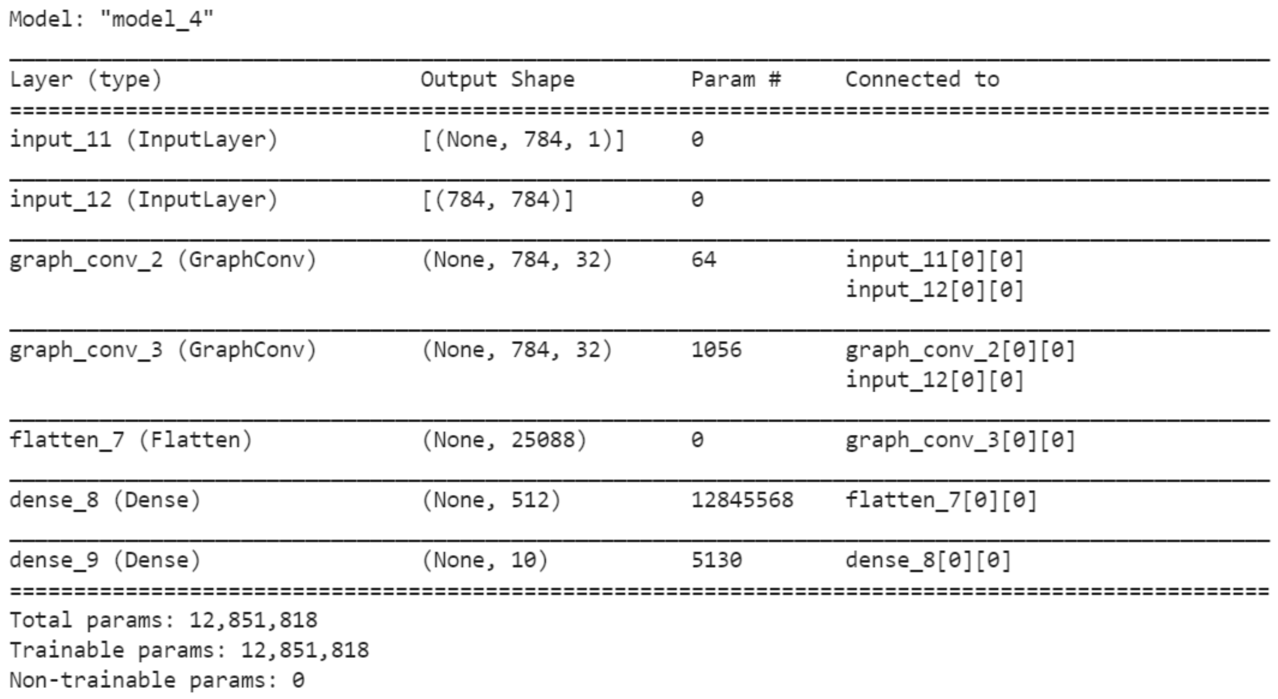


```
Evaluating model.
10000/10000 [==============================] - 1s 69us/sample - loss: 0.3612 - acc: 0.9035
Done.
Test loss: 0.36120691437721253
Test acc: 0.9035000205039978
```

Figure 5.12: Evaluation results of the fully connected model for MNIST dataset

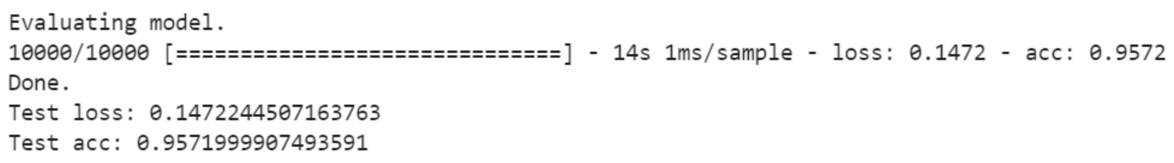
For the second part of the experiment, the Graph convolutional network for classification with different feature graphs is designed, and the model summary is shown in Table 5.6. As compared to the previous fully connected dense model, this model uses extra adjacency information as the convolution filter.

Table 5.6: Summary of GCN model for MNIST dataset



```
Model: "model_4"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_11 (InputLayer)           [(None, 784, 1)]     0
__________________________________________________________________________________________________
input_12 (InputLayer)           [(784, 784)]         0
__________________________________________________________________________________________________
graph_conv_2 (GraphConv)        (None, 784, 32)      64          input_11[0][0]
                                                                 input_12[0][0]
__________________________________________________________________________________________________
graph_conv_3 (GraphConv)        (None, 784, 32)      1056        graph_conv_2[0][0]
                                                                 input_12[0][0]
__________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 25088)        0           graph_conv_3[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 512)          12845568    flatten_7[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 10)           5130        dense_8[0][0]
==================================================================================================
Total params: 12,851,818
Trainable params: 12,851,818
Non-trainable params: 0
```

Then, this GCN model is trained with the same training and validation set. In Figure 5.13, the evaluation result of this model is shown with the same total loss and accuracy metrics.

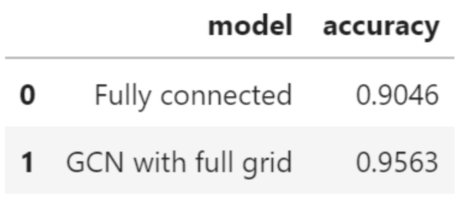


```
Evaluating model.
10000/10000 [==============================] - 14s 1ms/sample - loss: 0.1472 - acc: 0.9572
Done.
Test loss: 0.1472244507163763
Test acc: 0.9571999907493591
```

Figure 5.13: Evaluation results of GCN model for MNIST dataset

In other words, the images from the MNIST dataset are converted into the graph structure; an adjacency matrix is extracted using graph libraries. In the graph classification scenario, such information is used by the GCN model to classify an image(represented as graph data) into 10 digit classes. Hence, When full graph grid is used, the GCN model is performing better than the fully connected dense model, as shown in Table 5.7, and are able to process unstructured graph scenario successfully. Also, the current GCN model is still comparable to the popular CNN model, which has 99% of accuracy (Brownlee, 2019); this shows GCN can also handle non-structural data efficiently and covers more wide applications area.

Table 5.7: Comparision of models performance on MNIST dataset



	model	accuracy
0	Fully connected	0.9046
1	GCN with full grid	0.9563

Also, intermediate outputs of GCN layers can be visualized by using the method in the spektral library. Some of these intermediate outputs are plotted in Figure 5.7; these rearranged outputs also indicate learning of the GCN model is similar to standard Convolutional neural networks.

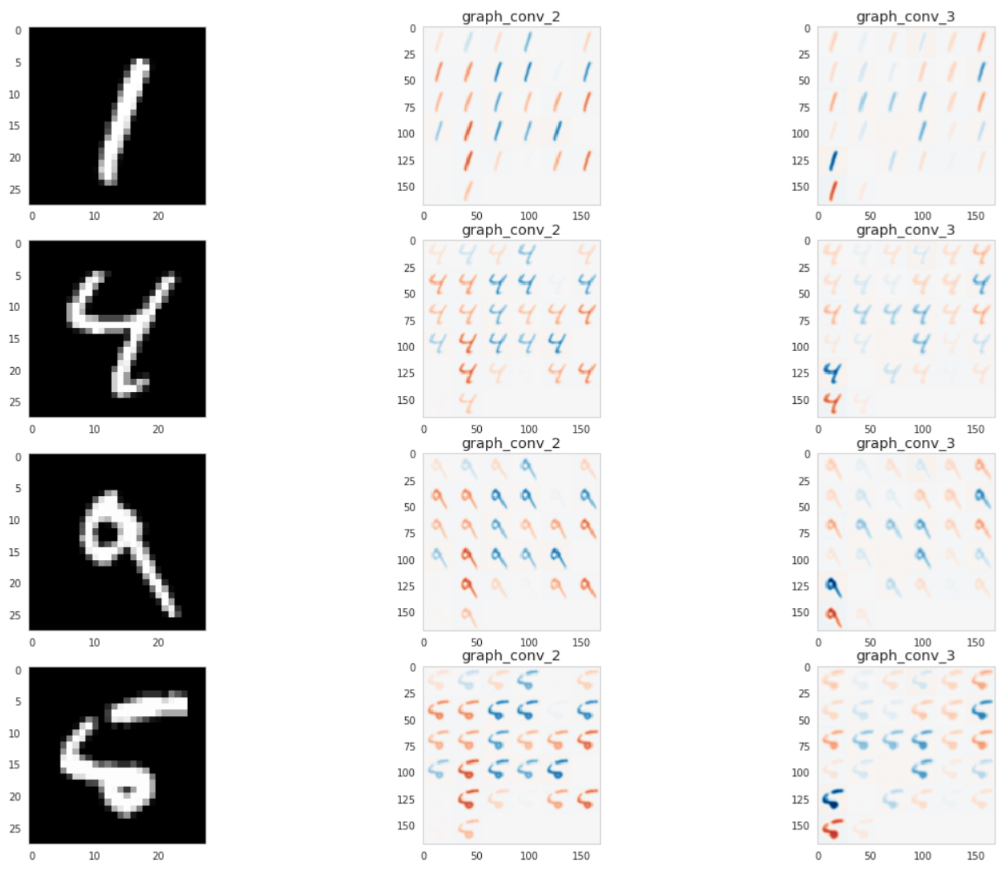


Figure 5.14: Intermediate output of GCN layers for MNIST dataset

5.3 Limitation

The approach taken to design the GCN models and for implementation in this work has some limitations. This section provides details about such limitations of this work.

In the first experiment performed on Zachary's karate club dataset, since the dataset does not contain a pre-defined network relationship between members/nodes, the comparison can't be performed on the closeness or argument between the club members.

Similar to most of the graph neural networks, GCN models used herein experiments are shallow in nature as compared to traditional deep neural networks that enhance expressive power. Also, over-smoothing can occur if many GCN layers are stacked together,i.e., all nodes can converge to similar values.

Graph datasets used in this work are static in nature and modeled accordingly. But the

GCN layer can't accept dynamically changing graph structures. The GCN model used here can't adapt to the appearance or disappearance of edges and nodes.

Although in this work, experiments are also performed on non-structured scenarios where datasets are not in graph format naturally, it can be construed that the methods utilized are not optimal to generate a graph from low-dimensional raw grids. This method that works on one type of dataset can't be generalized to another.

Also, GCN models used here can suffer in the case of embedding nodes for large web-scale scenarios such as a social network. In such scenarios, calculating graph Laplacian can become computational expensive because of millions of nodes and edges.

Chapter 6

Conclusion

The graph neural networks have become an efficient, widespread, and influential tool for machine learning applications in the graph domain. Even if GNNs models are not able to produce satisfactory results for many graph conditions, they achieved some remarkable success in different domains. This attention and progress are owing to advances in computational power, model design flexibility, and newly researched training methods.

In this work, the graph neural networks are introduced in details with some geometric deep learning models. The state-of-the-art GNN models are provided, and some GNN model variants are also presented based on applicable graph type and propagation steps used. Furthermore, GNNs are also categorized on training methodology and employed graph frameworks. With the perspective of application taxonomy, GNNs can be applied to both structural and non-structural scenarios. The design and limitations of the vanilla graph neural networks are explained, followed by a detailed discussion on specific GNN model for node and graph classification, called Graph Convolutional networks. Several graph libraries are presented that support to implement the GCN layers. For demonstrating the implementation of GCN layers, three datasets that are chosen under structural and non-structural scenarios are summarized, and the technical requirement for environment setup is provided. Complete implementation description for all three experiments with intermediate outputs and final result analysis are presented. Also, some limitations of the utilized GCN layer are addressed.

Finally, it can be construed that the graph neural networks are still a new research area getting lots of attention and advancing quite faster in recent years. The GNNs are targeted to analyze graphical data, but the application domain is not only limited to problems in graphs as GNN models can be generalized to any studies that can be modeled as graphs. Unlike the traditional “black box” machine learning algorithms that are trained only on features of training data and lack meaningful logic to perform, models based on the graph neural networks can also process the inherent logic and analyze a problem with more natural thinking.

6.1 Work Contribution and Challenges

This work can be used as the initial step to understanding the requirement of GNNs and a structured way to try into GNNs experiments. This work attempts to explore the recent researches in the domain of graph-based deep learning algorithms with supported graph libraries to design the GNN model and train them on selected datasets. Here, some basic mathematical formulation of nodes embedding is also discussed that can help to understand the background idea of a basic graph convolutional networks.

The following are some challenges are mentioned which are faced while working on the thesis:

1. Understanding the actual need of the graph neural network, that differs on application scenarios.
2. Difficulty in finding the basis for categorizing the graph neural network variants.
3. Selecting a graph library to be used for a particular experiment.
4. Designing the Spektral library-based GCN model on AWS Sagemaker notebook instances.

6.2 Future Work

In contrast to the number of limitations outlined in the previous section 5.3, there are some tasks that can be looked upon as further tasks.

- The selected GCN model can be tested and evaluated on some complex and more extensive datasets, which can help to determine the scalability of algorithms and whether it can be applied to practical scenarios.
- Since implemented models are shallow in nature, so designing a deep graph neural network can be significantly challenging work and will help in a better understanding of GNNs.
- For a more stable and adaptive GNN model, it should be able to handle the dynamic form of data, which will be a milestone in GNN related research works.
- Some more datasets can be supported by existing graph libraries, or more flexibility can be added into the GNN model designing method so that they can be easily customized to a broader range of applications.

References

Andreux, Mathieu, Emanuele Rodolà, Mathieu Aubry, and Daniel Cremers (Sept. 2014). "Anisotropic laplace-beltrami operators for shape analysis". DOI: 10.1007/978-3-319-16220-1_21 (cit. on p. 11).

Atwood, James and Don Towsley (2016). "Diffusion-convolutional neural networks". Curran Associates Inc., pp. 2001–2009 (cit. on p. 20).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). *Neural Machine Translation by Jointly Learning to Align and Translate.* URL: https://arxiv.org/pdf/1409.0473.pdf (cit. on p. 21).

Battaglia, Peter W, Jessica B Hamrick, et al. (2018). *Relational inductive biases, deep learning, and graph networks* (cit. on pp. 24, 39).

Battaglia, Peter W, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu (2016). *Interaction Networks for Learning about Objects, Relations and Physics.* arXiv.org. URL: https://arxiv.org/abs/1612.00222 (cit. on pp. 2, 25).

Beck, Daniel, Gholamreza Haffari, and Trevor Cohn (2018). *Graph-to-sequence learning using gated graph neural networks* (cit. on p. 18).

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". *IEEE Transactions on Neural Networks* 5.2, pp. 157–166 (cit. on p. 10).

Boaz Lee, John, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh (Nov. 2019). “Attention models in graphs”. *ACM Trans. Knowl. Discov. Data* 13.6. DOI: 10.1145/3363574. URL: https://doi.org/10.1145/3363574 (cit. on p. 14).

Boscaini, Davide, Jonathan Masci, Emanuele Rodolà, and Michael M Bronstein (2016). *Learning shape correspondence with anisotropic convolutional neural networks* (cit. on pp. 11, 13).

Bronstein, Michael M, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst (2017). “Geometric deep learning: Going beyond euclidean data”. *IEEE Signal Processing Magazine* 34.4, pp. 18–42. DOI: 10.1109/msp.2017.2693418. URL: http://dx.doi.org/10.1109/MSP.2017.2693418 (cit. on pp. 11, 14).

Brownlee, Jason (May 2019). *How to Develop a CNN for MNIST Handwritten Digit Classification.* Machine Learning Mastery. URL: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/ (cit. on p. 65).

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2013). *Spectral networks and locally connected networks on graphs* (cit. on pp. 12, 16).

Cheng, Jianpeng, Li Dong, and Mirella Lapata (Nov. 2016). “Long short-term memory-networks for machine reading”. Association for Computational Linguistics, pp. 551–561. DOI: 10.18653/v1/D16-1053. URL: https://www.aclweb.org/anthology/D16-1053 (cit. on p. 21).

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). *Learning phrase representations using RNN encoder-decoder for statistical machine translation* (cit. on p. 21).

Cho, Kyunghyun, van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (Oct. 2014). “Learning phrase representations using RNN encoder–decoder for statistical machine translation". Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://www.aclweb.org/anthology/D14-1179 (cit. on p. 10).

Chung, Fan (Dec. 1996). *Spectral Graph Theory*. Vol. 92. American Mathematical Society. URL: http://www.ams.org/books/cbms/092/ (cit. on p. 4).

club, Zachary karate (2013). *Zachary karate club*. Konect.cc. URL: http://konect.cc/networks/ucidata-zachary/ (cit. on p. 43).

Dai, Hanjun, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song (2017). "Learning combinatorial optimization algorithms over graphs". Curran Associates Inc., pp. 6351–6361 (cit. on p. 2).

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional neural networks on graphs with fast localized spectral filtering". Proceedings of the 30th International Conference on Neural Information Processing Systems. Curran Associates Inc., pp. 3844–3852 (cit. on pp. 12, 50).

docs.dgl.ai (2020). *Overview of DGL — DGL 0.4.3post2 documentation*. docs.dgl.ai. URL: https://docs.dgl.ai/index.html (cit. on pp. 37, 46).

Duvenaud, David, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams (2015). *Convolutional networks on graphs for learning molecular fingerprints* (cit. on p. 25).

Dwivedi, Vijay Prakash, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson (2020). *Benchmarking Graph Neural Networks*. arXiv.org. URL: https://arxiv.org/abs/2003.00982 (cit. on p. 24).

Fey, Matthias and Jan Eric Lenssen (2019). *Fast graph representation learning with PyTorch geometric* (cit. on p. 39).

Fouss, François, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens (July 2012). “An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification”. *Neural Networks* 31, pp. 53–72. DOI: 10.1016/j.neunet.2012.03.001 (cit. on p. 20).

Fout, Alex, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur (2017). “Protein interface prediction using graph convolutional networks”. Curran Associates Inc., pp. 6533–6542 (cit. on p. 2).

Gilmer, Justin, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl (2017). “Neural message passing for quantum chemistry”. JMLR.org, pp. 1263–1272 (cit. on p. 23).

Gori, M., G. Monfardini, and F. Scarselli (July 2005). *A new model for learning in graph domains*. IEEE Xplore. DOI: 10.1109/IJCNN.2005.1555942. URL: https://ieeexplore.ieee.org/document/1555942 (cit. on pp. 11, 27).

Goyal, Palash and Emilio Ferrara (2018). “Graph embedding techniques, applications, and performance: A survey”. *Knowledge-Based Systems* 151, pp. 78–94. DOI: https://doi.org/10.1016/j.knosys.2018.03.022. URL: http://www.sciencedirect.com/science/article/pii/S0950705118301540 (cit. on p. 4).

Grattarola, Daniele (2020). *spektral: A Python framework for relational representation learning*. PyPI. URL: https://pypi.org/project/spektral/0.0.11/ (cit. on p. 38).

Grattarola, Daniele and Cesare Alippi (2020). *Graph neural networks in TensorFlow and keras with spektral* (cit. on p. 38).

Grover, Aditya and Jure Leskovec (2016). “Node2vec: Scalable feature learning for networks”. Association for Computing Machinery, pp. 855–864. DOI: 10.1145/2939672.2939754. URL: https://doi.org/10.1145/2939672.2939754 (cit. on p. 5).

Hamaguchi, Takuo, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto (2018). “Knowledge Base Completion with Out-of-Knowledge-Base Entities: A Graph Neural Network Approach”. *Transactions of the Japanese Society for Artificial Intelligence* 33, F-H72$_1$ 10. DOI: 10.1527/tjsai.f-h72 (cit. on p. 2).

Hamilton, William L., Rex Ying, and Jure Leskovec (Apr. 2018). “Representation Learning on Graphs: Methods and Applications”. *arXiv:1709.05584 [cs]*. URL: https://arxiv.org/abs/1709.05584 (cit. on pp. 4, 5).

– (2017). “Inductive representation learning on large graphs”. Curran Associates Inc., pp. 1025–1035 (cit. on p. 22).

Hamilton, William L, Justine Zhang, Cristian Danescu-Niculescu-Mizil, Dan Jurafsky, and Jure Leskovec (2017). *Loyalty in online communities* (cit. on p. 2).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). *Deep residual learning for image recognition* (cit. on p. 22).

Henaff, Mikael, Joan Bruna, and Yann LeCun (2015). *Deep convolutional networks on graph-structured data* (cit. on p. 12).

Hochreiter, S, Y Bengio, P Frasconi, and J Schmidhuber (2001). “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”. Ed. by Kremer S C and Kolen J F. IEEE Press (cit. on p. 10).

Hochreiter, Sepp and Jürgen Schmidhuber (Dec. 1997). “Long short-term memory”. *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 21).

Jain, Ashesh, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena (2015). *Structural-rnn: Deep learning on spatio-temporal graphs* (cit. on p. 17).

Kampffmeyer, M., Y. Chen, X. Liang, H. Wang, Y. Zhang, and E P Xing (2019). “Rethinking knowledge graph propagation for zero-shot learning”, pp. 11479–11488 (cit. on p. 26).

Khamsi, Mohamed A. and William A. Kirk (Mar. 2001). *An Introduction to Metric Spaces and Fixed Point Theory*. John Wiley & Sons, Inc. DOI: 10.1002/9781118033074 (cit. on p. 30).

Khan, Muhammad Raza and Joshua E. Blumenstock (July 2019). “Multi-GCN: Graph Convolutional Networks for Multi-View Networks, with Applications to Global Poverty”. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, pp. 606–613. DOI: 10.1609/aaai.v33i01.3301606 (cit. on p. 18).

Kipf, Thomas and Max Welling (2016). *Semi-Supervised Classification with Graph Convolutional Networks* (cit. on pp. 2, 12, 20, 21, 33, 48, 50).

Koller, Daphne and Nir Friedman (2012). *Probabilistic graphical models principles and techniques*. Cambridge, Mass. Mit Press [Ca (cit. on p. 20).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on p. 10).

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86, pp. 2278–2324. DOI: 10.1109/5.726791 (cit. on p. 5).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (May 2015). “Deep learning”. *Nature* 521, pp. 436–444. DOI: 10.1038/nature14539 (cit. on p. 4).

LeCun, Yann and Corinna Cortes (2010). *MNIST handwritten digit database*. URL: http://yann.lecun.com/exdb/mnist/ (cit. on pp. 43, 45).

Li, Yaguang, Rose Yu, Cyrus Shahabi, and Yan Liu (2017). *Diffusion convolutional recurrent neural network: Data-driven traffic forecasting* (cit. on p. 17).

Li, Yujia, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel (2015). *Gated graph sequence neural networks* (cit. on p. 12).

Ma, Yao, Suhang Wang, Charu C Aggarwal, Dawei Yin, and Jiliang Tang (2018). *Multi-dimensional graph convolutional networks* (cit. on p. 18).

Masci, Jonathan, Davide Boscaini, Michael M Bronstein, and Pierre Vandergheynst (2015). *Geodesic convolutional neural networks on Riemannian manifolds* (cit. on pp. 11, 13).

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv.org. URL: https://arxiv.org/abs/1301.3781 (cit. on pp. 4, 5).

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). *Distributed representations of words and phrases and their compositionality* (cit. on p. 12).

Monti, Federico, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael Bronstein (July 2017). "Geometric deep learning on graphs and manifolds using mixture model CNNs", pp. 5425–5434. DOI: 10.1109/CVPR.2017.576 (cit. on p. 20).

Morel, J., B. Coll, and A. Buades (2005). "A non-local algorithm for image denoising". Vol. 3. IEEE Computer Society, pp. 60–65. DOI: 10.1109/CVPR.2005.38. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2005.38 (cit. on p. 14).

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena (2014). "DeepWalk: Online learning of social representations". Association for Computing Machinery, pp. 701–710. DOI:

10.1145/2623330.2623732. URL: https://doi.org/10.1145/2623330.2623732 (cit. on p. 4).

Sagemaker, AWS (2020). *Train a Deep Graph Network - Amazon SageMaker*. docs.aws.amazon.com. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/deep-graph-library.html (cit. on p. 37).

Sanchez-Gonzalez, Alvaro, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia (2018). *Graph networks as learnable physics engines for inference and control* (cit. on pp. 2, 23).

Sazli, Murat (Jan. 2006). "A brief review of feed-forward neural networks". *Communications, Faculty Of Science, University of Ankara* 50, pp. 11–17. DOI: 10.1501/0003168 (cit. on p. 9).

Scarselli, F., M. Gori, A C Tsoi, M. Hagenbuchner, and G. Monfardini (2009). "The graph neural network model". *IEEE Transactions on Neural Networks* 20.1, pp. 61–80 (cit. on pp. 11, 12, 27, 28, 29).

Schlichtkrull, Michael, Thomas Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling (2017). *Modeling Relational Data with Graph Convolutional Networks*. URL: https://arxiv.org/pdf/1703.06103.pdf (cit. on p. 18).

Sen, Prithviraj, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad (Sept. 2008). "Collective Classification in Network Data". *AI Magazine* 29, p. 93. DOI: 10.1609/aimag.v29i3.2157. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/2157 (cit. on pp. 43, 44).

Simonovsky, Martin and Nikos Komodakis (July 2017). *Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs*. IEEE Xplore. DOI: 10.1109/CVPR.2017.11. URL: https://ieeexplore.ieee.org/document/8099494/ (cit. on p. 23).

Simonyan, Karen and Andrew Zisserman (2014). *Very deep convolutional networks for large-scale image recognition* (cit. on p. 10).

Sinha, Ayan, Jing Bai, and Karthik Ramani (2016). “Deep learning 3D shape surfaces using geometry images”. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Springer International Publishing, pp. 223–240 (cit. on p. 11).

StellarGraph (2018). *Welcome to StellarGraph’s documentation! — StellarGraph 1.0.0rc1 documentation*. Readthedocs.io. URL: https://stellargraph.readthedocs.io/en/v1.0.0rc1/index.html# (cit. on pp. 39, 48).

Sun, Yiwei, Ngot Bui, Tsung-Yu Hsieh, and Vasant Honavar (Nov. 2018). “Multi-view Network Embedding via Graph Factorization Clustering and Co-regularized Multi-view Agreement”. *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. DOI: 10.1109/icdmw.2018.00145 (cit. on p. 18).

Szegedy, C., Wei Liu, Yangqing Jia, P. Sermanet, S Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). “Going deeper with convolutions”. CVPR, pp. 1–9 (cit. on p. 10).

Tang, Jian, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei (2015). “LINE: Large-scale information network embedding”. International World Wide Web Conferences Steering Committee, pp. 1067–1077. DOI: 10.1145/2736277.2741093. URL: https://doi.org/10.1145/2736277.2741093 (cit. on pp. 5, 12).

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio (2017). *Graph attention networks* (cit. on p. 21).

Wang, X., Y. Ye, and A. Gupta (2018). “Zero-shot recognition via semantic embeddings and knowledge graphs”, pp. 6857–6866 (cit. on p. 26).

Wang, Xiao, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu (2019). “Heterogeneous graph attention network”. Association for Computing Machinery, pp. 2022–2032. DOI: 10.1145/3308558.3313562. URL: https://doi.org/10.1145/3308558.3313562 (cit. on p. 16).

Wang, Xiaolong, Ross Girshick, Abhinav Gupta, and Kaiming He (2017). *Non-local neural networks* (cit. on pp. 14, 23).

Wang, Zichen (Mar. 2020). *Deep Learning on graphs: convolution is all you need.* Medium. URL: https://towardsdatascience.com/deep-learning-on-graphs-convolution-is-all-you-need-3c1cf8f1e715 (cit. on p. 3).

Wikipedia Contributors, Backpropagation (Apr. 2019). *Backpropagation.* Wikipedia. URL: https://en.wikipedia.org/wiki/Backpropagation (cit. on p. 8).

Wilson, Jenny (2012). *Manifolds The Definition of a Manifold and First Examples.* URL: http://www.math.lsa.umich.edu/~jchw/WOMPtalk-Manifolds.pdf (cit. on p. 11).

Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu (2020). “A comprehensive survey on graph neural networks”. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. DOI: 10.1109/tnnls.2020.2978386. URL: http://dx.doi.org/10.1109/TNNLS.2020.2978386 (cit. on p. 15).

Yang, Cheng, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang (2015). “Network representation learning with rich text information”. AAAI Press, pp. 2111–2117 (cit. on p. 5).

Zhang, Daokun, Jie Yin, Xingquan Zhu, and Chengqi Zhang (2018). “Network Representation Learning: A Survey”. *IEEE Transactions on Big Data* 6, pp. 3–28. DOI: 10.1109/tbdata.2018.2850013 (cit. on p. 4).

Zhang, Yizhou, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu (2018). “Deep collective classification in heterogeneous information networks”. International World Wide Web Conferences Steering Committee, pp. 399–408. DOI: 10.1145/3178876.3186106. URL: https://doi.org/10.1145/3178876.3186106 (cit. on pp. 15, 16).

Zhuang, Chenyi and Qiang Ma (2018). “Dual graph convolutional networks for graph-based semi-supervised classification”. International World Wide Web Conferences Steering Committee, pp. 499–508. DOI: 10.1145/3178876.3186116. URL: https://doi.org/10.1145/3178876.3186116 (cit. on p. 21).

Appendix A

- The Implemented codes under this work, along with all used papers links, intermediate, results and deployment instructions are available at the following link GitHub repository:

  https://github.com/vishalkumarmishra7/GNN_Investigation_TCD_Dissertation

  A copy of presentation is also updated here for reference.

- Karate Epochs and loss value while training model:

```
Epoch 0 | Loss: 0.7272
Epoch 1 | Loss: 0.7049
Epoch 2 | Loss: 0.6840
Epoch 3 | Loss: 0.6638
Epoch 4 | Loss: 0.6441
Epoch 5 | Loss: 0.6255
Epoch 6 | Loss: 0.6072
Epoch 7 | Loss: 0.5893
Epoch 8 | Loss: 0.5701
Epoch 9 | Loss: 0.5494
Epoch 10 | Loss: 0.5282
--------------------------------
Epoch 27 | Loss: 0.0738
Epoch 28 | Loss: 0.0598
Epoch 34 | Loss: 0.0162
Epoch 35 | Loss: 0.0131
Epoch 36 | Loss: 0.0107
Epoch 37 | Loss: 0.0087
Epoch 38 | Loss: 0.0072
----------------------------
```

```
Epoch 54 | Loss: 0.0009
Epoch 55 | Loss: 0.0009
Epoch 56 | Loss: 0.0008
Epoch 60 | Loss: 0.0007
Epoch 61 | Loss: 0.0006
Epoch 62 | Loss: 0.0006
Epoch 63 | Loss: 0.0006
Epoch 64 | Loss: 0.0006
----------------------
Epoch 87 | Loss: 0.0004
Epoch 94 | Loss: 0.0003
Epoch 95 | Loss: 0.0003
Epoch 96 | Loss: 0.0003
Epoch 97 | Loss: 0.0003
Epoch 98 | Loss: 0.0003
Epoch 99 | Loss: 0.0003
```

Acronyms

ACNN	Anisotropic Convolutional Neural Network. 13
CNN	Convolutional Neural Network. III
DCNN	diffusion-convolutional neural network. 20
DCRNN	Diffusion Convolutional Recurrent Neural Network. 17
DNN	Deep Neural Network. III
ECC	Edge-Conditioned Convolution. 23
FCNN	Convolutional Neural Network. 3
GAT	Graph Attention Network. 21
GCN	Graph Convolutional Netwrok. III
GCNN	Geodesic Convolutional Neural Networks. 13
GDL	Geometric deep learning. 11
GGNN	Dual Graph Convolutional Neural Network. 21
GLCN	Graph Learning Convolutional Network. 20
GN	Graph Network. 23
GNN	Graph Neural Network. III
GPUs	Graphics Processing Units. 2

GRU	Gated Recurrent Unit. 3
LINE	Convolutional Neural Network. 5
LSTM	Long Short-Term Memory. III
MoNet	Mixture model networks. 20
MPNN	Message Passing Neural Network. 23
MVNE	Multi-View Network Embedding. 18
NLNN	Non-Local Neural Network. 23, 24
R-GCNs	Relational Graph Convolutional Networks. 18
RNN	Recurrent Neural Network. III
SVM	Support Vector Machine. 3
TADW	Text-Associated DeepWalk. 5
TNSE	t-Distributed Stochastic Neighbor Embedding. 49