الجمهورية الجزائرية الديمقراطية الشعبية

People's Democratic Republic of Algeria

وزارة التعليم العالي و البحث العلمي

Ministry of Higher Education and Scientific Research

Registration n°: ...../...../...../...../.....

جامعة غرداية

University of Ghardaia

كلية العلوم والتكنولوجيا

Faculty of Science and Technology

قسم الرياضيات والاعلام الي

Department of Mathematics and Computer Science

مخبر الرياضيات والعلوم التطبيقية

Mathematics and Applied Sciences Laboratory

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master

Domain: Mathematics and Computer

Field: Computer Science

Specialty: Intelligent Systems for Knowledge Extraction

Topic

Autoencoder Based Community Detection

Kaoutar Zita

Publicly defended on June 24, 2024

Jury members:

Ahmed Saidi	MCB	Univ.Ghardaia	President
Asma Bouchekouf	MAA	Univ.Ghardaia	Examiner
Slimane Bellaouar	MCA	Univ.Ghardaia	Supervisor
Abdelfateh Bekkair	PhD student	Univ.Ghardaia	Co-Supervisor

Academic Year: 2023/2024

Acknowledgment

First and foremost, I thank **Almighty God** for guiding my path and giving me strength for this work.

I extend my deepest appreciation to my supervisor, **Dr.Slimane Bellaouar**, for his unwavering support, patience, and invaluable guidance throughout the research and writing of this thesis.

I am profoundly thankful to my co-supervisor, **Abdelfateh Bekkair**, for his exceptional dedication and unwavering commitment, which have been pivotal in every phase of this endeavor.

Special acknowledgment is extended to **Dr.Slimane Oulad Naoui** for his valuable advice and support throughout this process.

My heartfelt thanks go to all those who have supported me, whether near or far, in accomplishing this endeavor.

بِسْمِ اللَّهِ الرَّحْمَانِ الرَّحِيمِ

الْحَمْدُ لِلَّهِ الَّذِي أَنْزَلَ عَلَى عَبْدِهِ الْكِتَابَ وَلَمْ يَجْعَلْ لَهُ عِوَجًا (١) قَيِّمًا لِيُنْذِرَ بَأْسًا شَدِيدًا مِنْ لَدُنْهُ وَيُبَشِّرَ الْمُؤْمِنِينَ الَّذِينَ يَعْمَلُونَ الصَّالِحَاتِ أَنَّ لَهُمْ أَجْرًا حَسَنًا (٢)

[سورة الكهف]

إلا ما عاش فينا قبل أن نعيش فيه، وعرفناه في دفاتر التضحيات إلى وطننا الثاني فلسطين، قبلتنا الأولى ومسرى حبيبنا ونبينا الكريم جمعنا الله في أقصاها فاتحين مهللين مكبرين، وليس ذلك على اللّه بعسير .

إلى من قال فيهما اللّه عز وجل

وَاخْفِضْ لَهُمَا جَنَاحَ الذُّلِّ مِنَ الرَّحْمَةِ وَقُل رَّبِّ ارْحَمْهُمَا كَمَا رَبَّيَانِي صَغِيرًا

[سورة الإسراء: ٢٤]

إلى من أحمل إسمه بكل فخر إلى من دعمني منذ الصغر و آنار دربي لتحقيق حلمي إلى من نال منه التعب و تحمل قساوة الحياة لأجلنا إلى الذي قال لي يوما لن تشقي ما دمت حيا إلى سندي و قوتي أماني و مأمني ... أبي الغالي أطال اللّه في عمره .

إلى التي لا يطيب النهار إلا برؤيتها و لا تحلو الأيام إلا بوجودها إلى التي حملتني وهنا على وهن و إلى من كان دعاؤها سر نجاحي و حنانها بلسم جراحي ستبقى كلماتك نجوم أهتدى بها اليوم و في الغد و إلى الأبد ... أمي الغالية أطال اللّه في عمرها .

إلى الذين قال فيهم اللّه

سَنَشُدُّ عَضُدَكَ بِأَخِيكَ

إلى من انتظروا قطاف ثمرة جهدي طويلاً، فكانوا شركاء كل بسمة ودمعة وحسرة ... أحباب قلبي، أخوتي الأعزاء دمتم لي سندًا.

إلى من كان لي صديقًا و قريبًا و سندًا والتي كانت كلماته تواسيني لكي أنهض من جديد

وصال وخيرة .

إلى كل من نسيهم قلمي و تذكرهم قلبي.

مـلـخـص

اكتشاف المجتمع أمر بالغ الأهمية للكشف عن البُنى الفرعية المتماسكة داخل الأنظمة المعقدة. تقدم هذه المجتمعات رؤى حول مجموعات الكيانات المترابطة، مما يمكن أن يكون ذا قيمة خاصة في مجالات مثل تحليل الشبكات الاجتماعية، واسترجاع المعلومات، والمكتبات الببليومترية. في هذه الدراسة، نقترح تصنيفًا لأساليب اكتشاف المجتمع بناءً على مشفرات الرسوم البيانية الذاتية (GAEs)، ونصنفها إلى نماذج مشفر بسيط ونماذج مزدوجة المشفر. نجري تحليلًا مقارنًا لهاتين الفئتين، مع التركيز على نوع بنية المشفر وتقييم أدائها على الشبكات الحقيقية. للتقييم الدقيق، نستخدم مقاييس NMI و ARI و F1. بالإضافة إلى ذلك، نفحص كفاءة وقت التشغيل لكل نموذج استنادًا إلى العصور. تشير النتائج إلى أن النماذج مزدوجة المشفر، لا سيما تلك التي تحتوي على آليات الانتباه، تظهر أداءً متفوقًا عمومًا، خاصة في مجموعات البيانات المعقدة، رغم المتطلبات الحسابية الأعلى. تؤكد هذه النتائج على إمكانات النماذج مزدوجة المشفر في مهام تحليل الشبكات المتقدمة. تشمل التوصيات المستقبلية دراسة تصميمات الشبكات العصبية الأكثر تقدمًا وتأثير عوامل النمذجة وإعداد البيانات على اكتشاف المجتمع عبر مختلف المجالات.

كلمات مفتاحية: كشف المجتمعات، مشفر الرسوم البيانية التلقائي، الشبكات الموصوفة، مشفر بسيط، مشفر مزدوج.

Abstract

Community detection is crucial for uncovering cohesive substructures within complex systems. These communities provide insights into clusters of interconnected entities, which can be particularly valuable in various domains such as social network analysis, information retrieval, and bibliometrics. In this study, we propose a taxonomy of community detection methods based on graph autoencoders (GAEs), categorizing them into simple encoder and dual encoder models. We conduct a comparative analysis of these two categories, focusing on the type of encoder architecture and assessing their performance on real networks. For a more precise evaluation, we use NMI, ARI, and F1-measure as evaluation metrics. Additionally, we examine the running time efficiency of each model based on epochs. The findings indicate that dual encoder models, especially those with attention mechanisms, generally exhibit superior performance, particularly in complex datasets, despite higher computational demands. These results underscore the potential of dual encoder models in advanced network analysis tasks. Future recommendations include examining more advanced neural network designs and the impact of modeling and data preparation factors on community detection across various domains.

Keywords: Community detection, Graph Autoencoder, Attributed networks, Simple encoder, Dual encoder.

Résumé

La détection des communautés est cruciale pour découvrir des sous-structures cohésives au sein de systèmes complexes. Ces communautés fournissent des informations sur les clusters d'entités interconnectées, ce qui peut être particulièrement précieux dans divers domaines tels que l'analyse des réseaux sociaux, la recherche d'informations et la bibliométrie. Dans cette étude, nous proposons une taxonomie des méthodes de détection de communautés basée sur des autoencodeurs de graphes (GAEs), les classant en modèles à encodeur simple et modèles à encodeur double. Nous réalisons une analyse comparative de ces deux catégories, en nous concentrant sur le type d'architecture d'encodeur et en évaluant leur performance sur des réseaux réels. Pour une évaluation plus précise, nous utilisons la NMI, l'ARI et la F1-mesure comme métriques d'évaluation. De plus, nous examinons l'efficacité du temps d'exécution de chaque modèle en fonction des époques. Les résultats indiquent que les modèles à encodeur double, en particulier ceux avec des mécanismes d'attention, affichent généralement une performance supérieure, notamment dans les ensembles de données complexes, malgré des exigences computationnelles plus élevées. Ces résultats soulignent le potentiel des modèles à encodeur double dans les tâches avancées d'analyse de réseaux. Les recommandations futures incluent l'examen de conceptions de réseaux neuronaux plus avancées et l'impact des facteurs de modélisation et de préparation des données sur la détection des communautés dans divers domaines.

Mots clés: Détection de communautés, Autoencodeur de graphe, Réseaux attribués, Encodeurs simples, Encodeurs doubles.

Contents

List of Figures **xi**

List of Tables **xii**

List of Acronyms **xiii**

1 Introduction **1**

Introduction **1**

1.1 Context and Motivations 1

1.2 Objectives 2

1.3 Thesis Organization 2

2 Basic Concepts **4**

2.1 Graph Theory Fundamentals 4

2.1.1 History of Graph Theory 5

2.1.2 Definitions 6

2.1.3 Graph types 7

2.1.4 Graph representations 11

2.1.5 Graph algorithms 13

2.1.6 Exploring Graph Theory and Network Applications 15

2.2 Community detection 18

2.2.1 Communities 18

2.2.2 Community detection 19

2.2.3 Static Community Detection 20

2.2.4 Dynamic Community Detection 20

2.3 Machine learning 21

2.3.1 Machine learning Types 21

2.3.2 Artificial Neural Networks 24

2.4 Graph Neural Networks 25

2.4.1 Message Passing Framework 26

2.4.2 Graph Neural Networks Types 29

2.4.3 Graph Neural Networks Applications 30

2.5 Autoencoder Technology 31

2.5.1 Different Autoencoder architecture 31

2.5.2 Autoencoders Applications 35

2.6 Conclusion 36

3 State Of The Art **37**

3.1 Traditional Methods 37

3.1.1 Hierarchical Clustering Methods 39

3.1.2 Modularity Optimization Methods 41
3.1.3 Dynamical Methods 43
3.1.4 Comparison between Traditional Methods 47
3.2 Deep Learning Methods 48
3.2.1 Convolutional Based Community Detection 49
3.2.2 Autoencoder-Based Community Detection 51
3.3 Conclusion 58

4 Experimental Study **59**
4.1 Implementation setup 59
4.1.1 Datasets 59
4.1.2 Environment 60
4.1.3 Evaluation Metrics 62
4.2 Implementation 64
4.3 Results and Discussion 67
4.4 Conclusion 70

Conclusion and Perspectives **71**

List of Figures

2.1 Königsberg in 1652 5

2.2 Simple graph Representation of order 6 6

2.3 Subgraphs. 7

2.4 Multigraph, Digraph, and Pseudograph Representations. 8

2.5 Weighted graph. 9

2.6 Complete graphs. 9

2.7 Two 3-partite graphs. 9

2.8 bipartite graph. 10

2.9 Planar and Non-Planar Graph Representation. 10

2.10 The Trees on Six Vertices. 11

2.11 Two Representations of an Undirected Graph. 12

2.12 Two Representations of a directed Graph. 12

2.13 Visualization of a simple social network 16

2.14 Visualization of a Citation Network 16

2.15 Visualization of food Web Networks 17

2.16 Protein-Protein Interaction Networks 17

2.17 Visualization of a Network with 3 Distinct Communities. 19

2.18 Six basic Operations of Community Evolution in Dynamic Networks . . 21

2.19 Machine Learning Types. 24

2.20 Biological neuron vs Artificial neuron 25

2.21 Message Aggregation in Two-Layer GNN 27

2.22 Autoencoder Architectur. 32

2.23 Convolutional Autoencoder Architecture 33

2.24 Variational Autoencoder Architecture. 34

3.1 Traditional and deep learning community detection methods. 38

3.2 Hierarchical Clustering Steps represented by Dendrogram 39

3.3 Louvain Algorithm Steps. 43

3.4 Architecture of Embedding Graph Auto-Encoder 53

3.5 Architecture of Adversarially Regularized Graph Autoencoder 54

3.6 Deep Attentional Graph Clustering 56

3.7 Deep Dual Graph attention Auto-Encoder architecture 57

4.1 Models performance comparison across datasets in terms of execution time per epoch. 69

List of Tables

2.1 Activation Functions. 25

3.1 Summary of Traditional Algorithms for Community Detection 48

3.2 Summary of GAE-Based Community Detection. 58

4.1 Statistical Summary of Citation Network Datasets 60

4.2 Models Performance on Cora Dataset 68

4.3 Models Performance on CiteSeer Dataset 68

4.4 Models Performance on PubMed Dataset 68

List of Acronyms

ANN	Artificial Neural Network
CAE	Convolutional Autoencoder
CNN	Convolutional Neural Network
DAE	Denoising Autoencoder
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GraphSAGE	Graph Sample and Aggregated
MLP	Multi-Layer Perceptron
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
SAE	Sparse Autoencoder
VAE	Variational Autoencoder
LPA	Label Propagation Algorithm
InfoMap	Information Map Algorithm
ARGAE	Adversarially Regularized Graph Autoencoder
ARVGA	Variational Adversarially Regularized Graph Autoencoder
DAEGC	Deep Attentional Embedded Graph Clustering
EGAE	Embedding Graph Auto-Encoder
DDGAE	Deep Dual Graph Attention Auto-Encoder

NMI	Normalized Mutual Information
ARI	Adjusted Rand Index

Chapter 1

Introduction

1.1 Context and Motivations

The investigation of networks constitutes a significant area within the broader framework of complex systems. Networks serve as a means to examine the intricate relationships inherent within diverse systems, including social, biological, and others. In this paradigm, nodes represent the fundamental elements of a system, while links denote the interactions between them.Within these networks, complex systems often exhibit a modular organization, wherein distinct compartments, known as communities. These communities are characterized by dense internal connections, distinguishing them from the sparser connections between compartments.

The identification of communities, known as community detection, is instrumental in unraveling the underlying organizational structures of networks. However, the lack of a universally accepted definition of what constitutes a community poses a notable challenge in this field. To address this challenge, numerous methods and algorithms have been proposed, each offering distinct approaches to uncovering communities in networks. Notable among these are hierarchical methods like the GN algorithm [18], dynamic methods such as the label propagation algorithm (LPA) [44], and algorithms like InfoMap [46] . Moreover, optimization techniques, as demonstrated by Louvain algorithm [5], aim to enhance the precision of community detection by maximizing modularity.

Despite their contributions, traditional community detection methods confront challenges persist, particularly with the increasing accumulation of data and the emergence of large-scale networks. These limitations highlight the need for innovative approaches that can address these issues more effectively. In response, emerging methodologies, such as deep learning-based community detection algorithms, have garnered attention. Among these, autoencoder architectures have emerged as promising alternatives, leveraging deep learning techniques to potentially overcome these challenges and extract more accurate community structures from network data.

1.2 Objectives

This project aims to conduct a comprehensive comparative study of various autoencoder architectures, specifically tailored to the challenge of community detection within complex networks. To achieve this, we propose a taxonomy of community detection methods based on graph autoencoders (GAEs), categorizing them into two main categories: GAE-Based Simple Encoder and GAE-Based Dual Encoder. Furthermore, within each of these categories, we classify the approaches based on whether they utilize Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs). The primary objective is to investigate and comprehend the applicability of different autoencoder architectures in this context, aiming to discern their respective strengths and weaknesses.

1.3 Thesis Organization

This thesis comprises three chapters, each focusing on distinct aspects of community detection in complex networks.

Chapter 2 In this chapter, we delve into the foundational principles of graph theory and its practical applications in various network domains like social networks, citation networks, and biological networks. We introduce the concept of community detection, covering both static and dynamic methods. Furthermore, we explore the intersection of graph theory and machine learning, discussing different types of machine learning and the emergence of Graph Neural Networks (GNNs) for network analysis. Finally, we investigate autoencoder technology, detailing its architectures and applications in network analysis.

Chapter 3 This chapter provides a comprehensive survey of community detection methods. We explore the evolution of community detection approaches, from traditional methodologies to advanced deep learning techniques, with a particular focus on autoencoder-based community detection.

Chapter4 In this chapter, we conduct a comparative analysis of graph autoencoder models for community detection. We distinguish between simple and dual encoders and evaluate their performance across various datasets. We detail the experimental setup, methodologies, and comparative results to assess the efficacy of each encoder type in identifying community structures.

As we delve into the realm of network analysis, our focus lies on the pivotal task of community detection. Through a comprehensive exploration of graph autoencoder models, we aim to uncover optimal strategies for delineating community structures within complex networks. Our journey through this study promises to unveil valuable insights, guiding future advancements in network analysis methodologies.

Chapter 2

Basic Concepts

Graph theory is all about understanding the connections between objects and offers a solid foundation for modeling diverse complex networks, such as social and biological networks. Understanding the structure and dynamics of these networks is crucial for applications like community detection.

In this chapter, we explore the basics of complex networks and their close connection to graph theory. Additionally, our inquiry extends beyond these foundational notions as we venture into the realm of Graph Neural Networks (GNNs), a potent and sophisticated tool that has emerged as a cornerstone in the field of network analysis.

We introduce autoencoders, a powerful deep learning technique, to complement existing methods for community detection in complex networks. By exploiting autoencoder architecture, we aim to achieve a deeper understanding of network structures and uncover hidden communities.

2.1 Graph Theory Fundamentals

This part establishes a foundation in graph theory by examining its historical development, fundamental terminology, and multifaceted applications. We elucidate the significance of graph algorithms and their tangible implications in real-world networks encompassing social media, citation analysis, and biological systems.

2.1.1 History of Graph Theory

Graph theory originated in 1736 with the groundbreaking work of Swiss mathematician Leonhard Euler (1707–1783). Euler's exploration of the famous ”Seven Bridges of Königsberg” problem serves as the foundation of graph theory [27].

The city of Königsberg, situated in Prussia (now Kaliningrad, Russia), boasted two significant islands, Kneiphof and Lomse, interconnected by seven bridges, including connections to the city's mainland. The intriguing puzzle revolved around discovering a path through the city, ensuring that each bridge was crossed precisely once.

Euler's brilliance resided in his ability to recognise that the four unique land areas and seven bridges were the most important aspects of the problem. He presented the problem with the first known graphic representation of a modern network. A modern graph consists of a collection of points known as vertices or nodes, joined by a series of lines known as edges. [27].

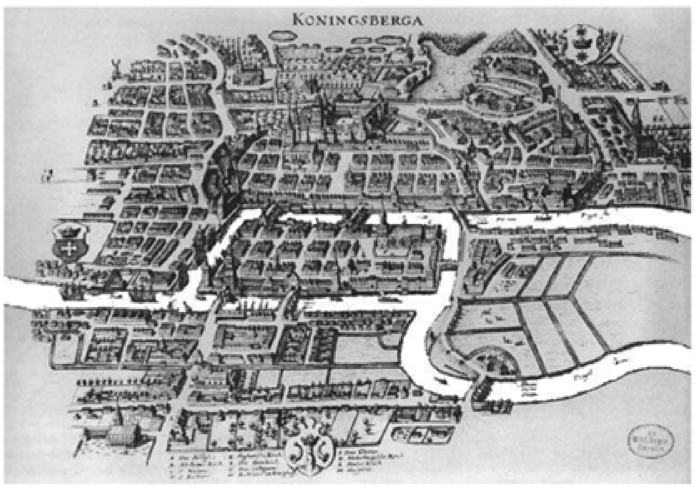


Figure 2.1: Königsberg in 1652[37].

2.1.2 Definitions

Graph: A graph G is defined as a pair of vertices V and edges E linking them. $E \subseteq V \times V$ implies that each edge in E is a subset of the Cartesian product of V. In finite graphs, the number of vertices and edges is indicated by $n = |V|$ (also known as graph order) and $m = |E|$ [10].

Vertex (or Vertices): Also known as nodes, vertices are the points or objects represented in the graph. Each vertex in the graph represents an individual element or entity [27].

Edge: Edges are the connections between pairs of vertices. An edge $e = \{a, b\}$ represents a link between vertices a and b [27].

Degree of a Vertex: The degree of a vertex v in a graph G is the count of edges that have v as one of their endpoints [7].

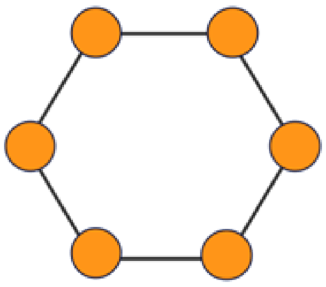


Figure 2.2: Simple graph Representation of order 6 .

Walk: A walk in G is a finite non-null sequence $W = v_1, e_1, v_2, e_2, v_3, e_3, \ldots, e_k, v_k$ whose terms are alternately vertices and edges (possibly revisiting vertices and using the same edge multiple times) [7].

Trail: A trail is a type of walk in a graph that visits each edge at most once [7].

Path: A path is a type of walk that visits each vertex and each edge in the graph exactly once [7].

Distance: In a graph G, the distance between two vertices x and y, denoted as $d_G(x,y)$, is the minimum number of edges in any path connecting x to y.[14].

Diameter of G**,** denoted by diam G, is the greatest distance between any pair of vertices in G[14].

Subgraphs: A graph H is a subgraph of a graph G, denoted as $H \subseteq G$, if the vertex set and edge set of H are subsets of the vertex set and edge set of G, respectively. When H is not identical to G, it is referred to as a proper subgraph, written as $H \subset G$.

An **induced subgraph** is a subgraph of G that contains a specific subset of vertices from G, as well as all edges that exist between those selected vertices. Finally, a **spanning subgraph** is a subgraph of G that contains all the vertices in the original graph [7].

Figure 2.3 illustrates an example of a subgraph within the graph G, including induced and spanning subgraphs.

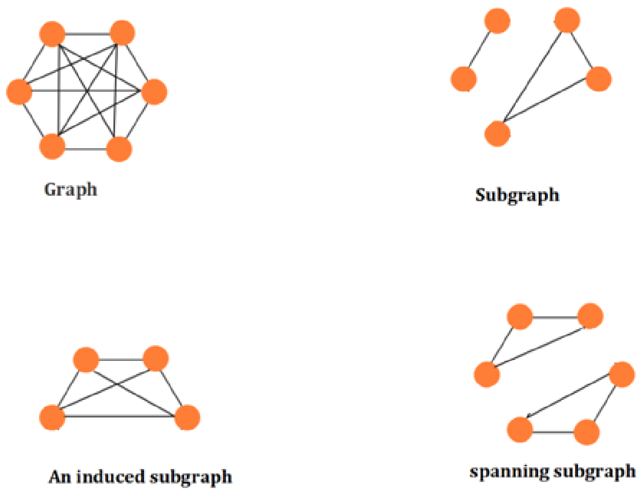


Figure 2.3: Subgraphs.

2.1.3 Graph types

There are indeed various types of graphs and graph-related concepts. We present a diverse range of graph types, each illustrating distinct structural characteristics and relationships within the field of graph theory.

Multigraph: is a graph where each edge $e \in E$ is associated with a pair of vertices $(v, w) \in V \times V$, and there can exist multiple edges that connect the same pair of vertices [21].

Directed Graph: A directed graph, also known as a digraph, is a type of graph where each edge has a direction associated with it. Such edges will be called arcs [21].

Pseudograph: A graph that allows edges to begin and end at the same vertex (loop) [21].

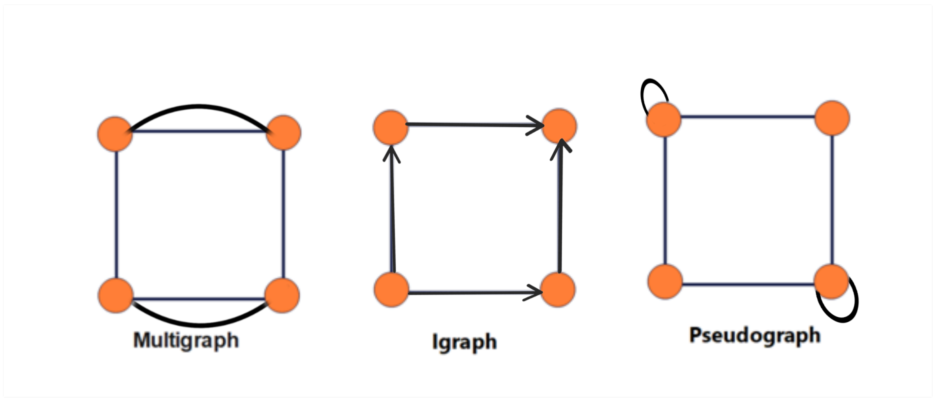


Figure 2.4: Multigraph, Digraph, and Pseudograph Representations.

Weighted graph: A graph denoted by a triple $G = (V, E, w)$, where V represents a collection of vertices, $E \subseteq V \times V$ represents a set of edges, and $w : E \to \mathbb{R}^+$ gives a non-negative weight to each edge $e \in E$ (as depicted in Figure 2.5). To simplify notation, $w(u, v) = 0$ if $(u, v) \notin E$.

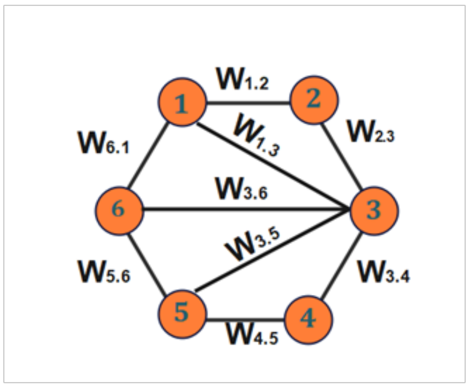


Figure 2.5: Weighted graph.

Complete Graph: A graph that contains an edge between all pairs of vertices. The complete graph of order p is denoted as K_p [21]

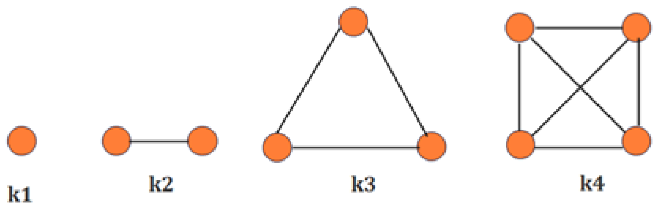


Figure 2.6: Complete graphs.

R-partite Graph: A graph is r-partite if its vertex set V can be divided into r classes $V_1, V_2, \ldots, V_r$ where no edge connects vertices within the same class[14].

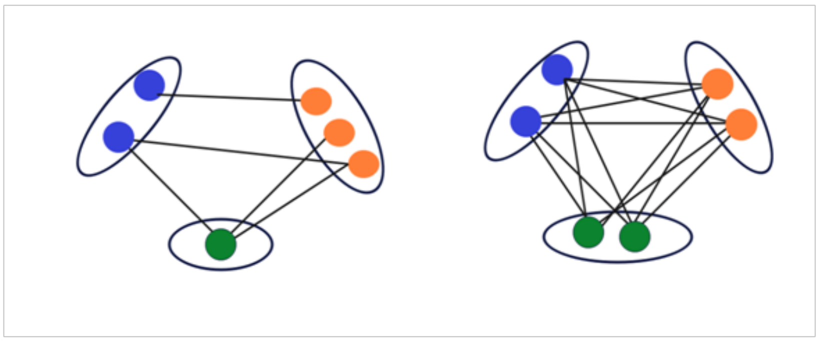


Figure 2.7: Two 3-partite graphs.

Bipartite Graph: is a specific type of graph that can be categorized as an r-partite graph with $r = 2$ [14].

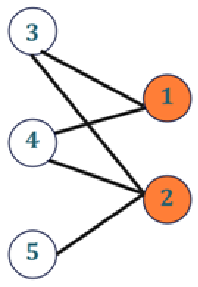


Figure 2.8: bipartite graph.

Planar Graph:A planar graph is a graph that may be formed on a plane so that its edges only overlap at their endpoints. [7].

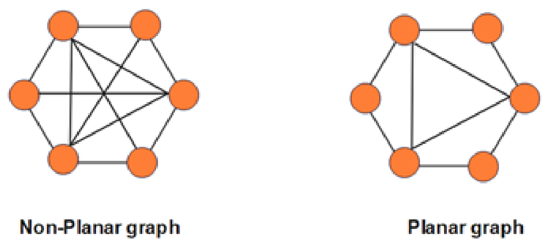


Figure 2.9: Planar and Non-Planar Graph Representation.

Cycle: A cycle is a sequence of vertices and edges that forms a closed loop, where the initial and final vertices are the same [10]. Formally, a path S is a cycle if:

$$S = \{(v, v_1), (v_1, v_2), \ldots, (v_{k-1}, v)\}$$

Circuit: A circuit is a directed cycle[10]. Formally, a path T is a circuit if:

$$T = \{(v \rightarrow v_1), (v_1 \rightarrow v_2), \ldots, (v_{k-1} \rightarrow v)\}$$

Acyclic Graph: An acyclic graph is a type of graph that does not contain any cycles [7].

Tree: A tree is a linked, acyclic graph. The figure depicts trees with six vertices. [7].

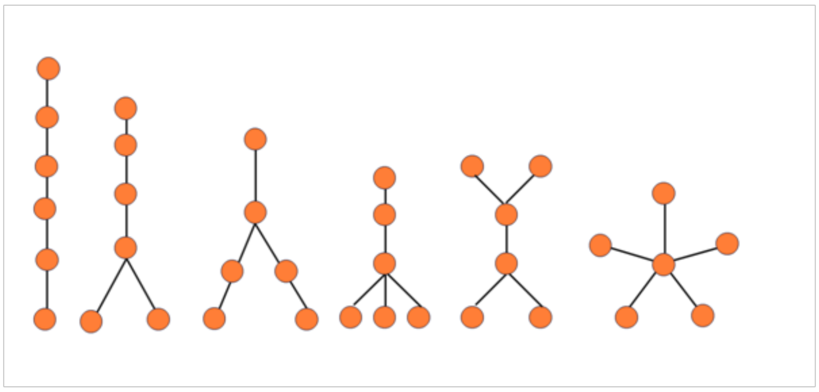


Figure 2.10: The Trees on Six Vertices.

2.1.4 Graph representations

When working with graphs in a computer program, it becomes essential to choose an appropriate representation method to efficiently store and manipulate the graph data. We will explore two standard ways of representing or maintaining a graph G in a computer's memory.

Adjacency Matrix

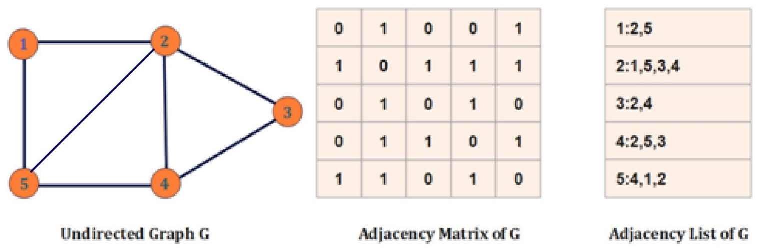
The adjacency matrix of a graph $G = (V, E)$ is a matrix $A = (a_{ij})$ of size $n \times n$, where n is the number of vertices in G. The element a_{ij} at the intersection of row i and column j represents the adjacency relationship between vertices i and j [51]. Each entry a_{ij} is defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Adjacency list

An adjacency list is a memory-based data structure utilized for graph representation. It comprises a series of lists or arrays, where each element represents a node in the graph. For directed graphs, each list contains the nodes directly reachable from the corresponding node, while for undirected graphs, each list contains neighboring nodes connected by edges.

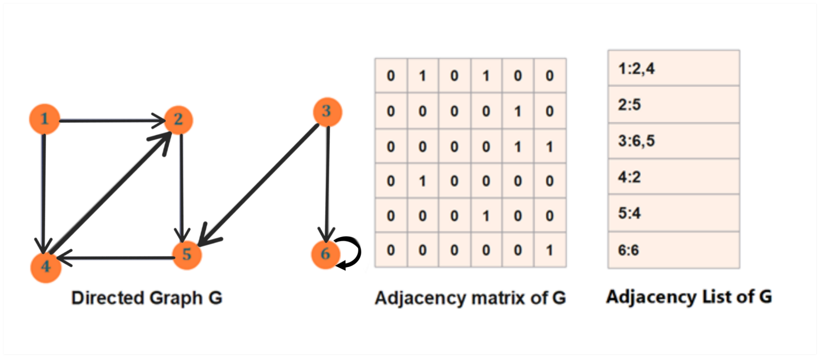
The list adjacency representation offers advantages over the sequential representation (adjacency matrix) by being more memory-efficient for sparse graphs and facilitating easier insertion and deletion of nodes and edges. It is widely used in graph algorithms and applications due to its flexibility and efficiency for various graph-related operations [51].



0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0

1:2,5
2:1,5,3,4
3:2,4
4:2,5,3
5:4,1,2

Figure 2.11: Two Representations of an Undirected Graph.



0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

1:2,4
2:5
3:6,5
4:2
5:4
6:6

Figure 2.12: Two Representations of a directed Graph.

2.1.5 Graph algorithms

Graphs are valuable tools for representing different problems, and there's a wide range of techniques to solve various issues. Below, we present a few of these methods.

Breadth-First Search (BFS) algorithm

Breadth-First Search is is a methodical graph traversal algorithm employed to systematically traverse the vertices and edges of a graph. It emphasizes visiting vertices at increasing distances from a selected source vertex, ensuring thorough exploration of the graph's structure.[49].

Algorithm 1 Breadth-First Search

```
procedure BFS($G$ : Graph, $s$ : Integer)
    $queue$ : Queue of Integer
    $marked$ : Array of Boolean
    $edgeTo$ : Array of Integer
    for $v$ in $G$.vertices do
        $marked[v] \leftarrow$ false
        $edgeTo[v] \leftarrow$ NULL
    end for
    $marked[s] \leftarrow$ true
    $queue$.enqueue($s$)
    while not $queue$.isEmpty() do
        $v \leftarrow queue$.dequeue()
        for $w$ in $G$.adj($v$) do
            if not $marked[w]$ then
                $edgeTo[w] \leftarrow v$
                $marked[w] \leftarrow$ true
                $queue$.enqueue($w$)
            end if
        end for
    end while
end procedure
```

The time complexity of BFS is denoted as $O(V + E)$, where V is the number of vertices and E is the number of edges in the graph. his complexity arises from the necessity to visit each vertex and edge at least once in the worst-case scenario.

Depth-First Search (DFS) algorithm

Depth-First Search (DFS) is a graph traversal algorithm used to systematically explore all the vertices and edges of a graph. It follows a systematic approach of visiting a vertex and then recursively exploring as far as possible along each branch before backtracking. This process continues until all vertices have been visited. DFS can be used to discover various properties of a graph, including connected components, cycles, and paths[49].

Algorithm 2 Depth-First Search

```
function DFS($G, v$)
    $marked[v] \leftarrow$ true
    $id[v] \leftarrow$ count
    for $w$ in $G$.adj($v$) do
        if not $marked[w]$ then
            DFS($G, w$)
        end if
    end for
end function
```

$O(V + E)$,each vertex and edge is visited at most once. The time complexity of visiting a vertex and processing its adjacent vertices is proportional to the number of adjacent vertices (the degree of the vertex). In the worst case, you visit every vertex and process all of its adjacent vertices.

Dijkstra's algorithm

Dijkstra is a method for determining the shortest pathways between nodes in a network that have non-negative edge weights. The algorithm assigns a distance label to each node, which initially reflects an upper bound on the shortest path length to that node. It works iteratively by growing the set of permanently labelled nodes, beginning with the source node. At each step, the algorithm selects the node with the least temporary label, makes it permanent, and updates the labels of its neighbouring nodes if a shorter path is discovered.[1].

Algorithm 3 Dijkstra's Algorithm

```
function DIJKSTRA(Graph, source)
    $S \leftarrow \emptyset$
    for each node $i \in$ Graph do
        $d(i) \leftarrow \infty$
    end for
    $d(\text{source}) \leftarrow 0$
    $\text{pred}(\text{source}) \leftarrow 0$
    while $|S| < n$ do
        $i \leftarrow$ node in Graph for which $d(i) = \min\{d(j) : j \in S\}$
        $S \leftarrow S \cup \{i\}$
        for each $(i, j) \in A(i)$ do
            if $d(j) > d(i) + cij$ then
                $d(j) \leftarrow d(i) + cij$
                $\text{pred}(j) \leftarrow i$
            end if
        end for
    end while
    return $d$
end function
```

Dijkstra's algorithm efficiently finds shortest paths in graphs, taking $O(n^2)$ time for dense networks due to unbalanced node selections and distance updates.

2.1.6 Exploring Graph Theory and Network Applications

Graphs are versatile data structures that are used to represent and analyze a wide variety of relationships and information in different fields.Here, we present some key common applications of graphs to underscore their significance.

Social network

Social networks is a representation of people and their connections. Each person is a node, and the links between nodes show real-world or online interactions. People can trust each other based on their interactions, like rating movie recommendations. Since everyone interacts with many others, the network becomes complex[30].

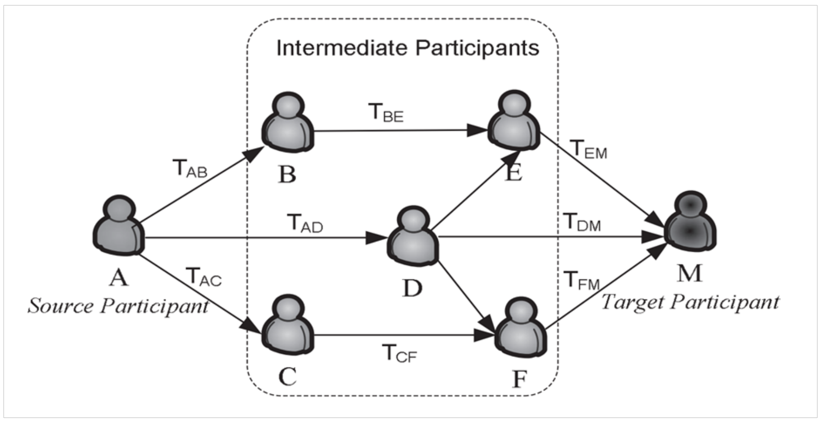


Figure 2.13: Visualization of a simple social network [30].

Citation Networks

Citation networks are like academic trails connecting research papers. When a paper mentions or cites another paper, a link is formed between them. These links create a network where papers are connected based on who references whom[15].

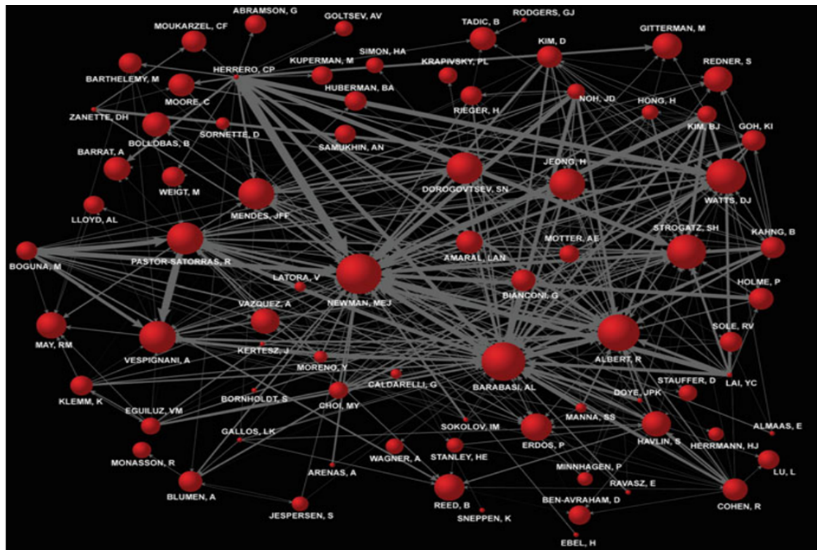


Figure 2.14: Visualization of a Citation Network[43].

Biological networks

Networks play a significant role in various aspects of biology, helping scientists and researchers better understand complex biological systems. We focus on protein-protein interactions (PPI) and food webs, but it's important to note that there are numerous other valuable applications of networks in this field as well.

- **Ecological food Web Networks** Web food networks portray the interconnected relationships between species in a specific environment based on who eats whom. These networks encompass a wide range of organisms, from plants to bacteria and animals. Links in these networks show how energy and nutrients move through different trophic levels, including herbivores, predators, and parasites. At the core of every food web are autotrophs, like plants, that produce their own energy[16].

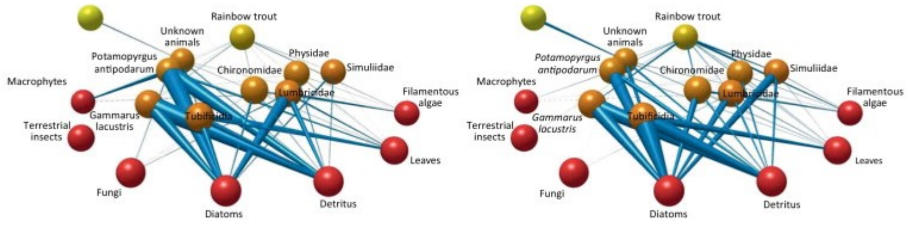


Figure 2.15: Visualization of food Web Networks [57].

- **Protein networks** are intricate structures resembling vast interconnected webs. They consist of components from genomic regulatory systems forming vertices, linked by directed edges that signify regulatory relationships. Key to gene function is protein-protein interactions, where proteins act as vertices connected by directed edges representing pairwise interactions. These networks can display both bidirectional connections and self-interactions, resembling food webs [15].

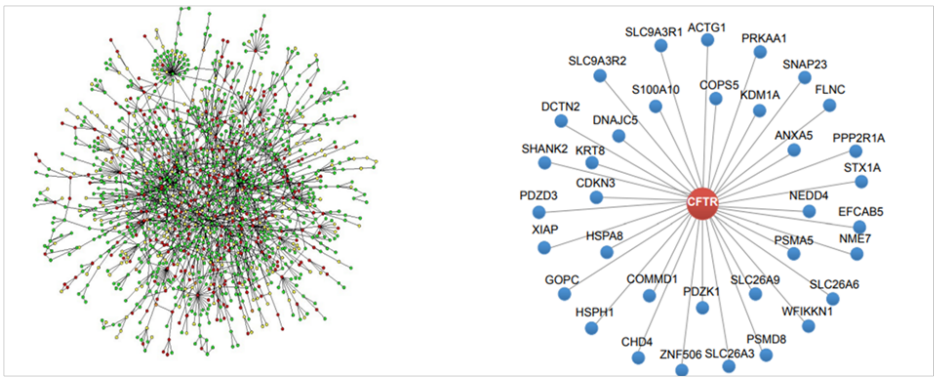


Figure 2.16: Protein-Protein Interaction Networks[8].

2.2 Community detection

In this section, we dive into the realm of community detection, unveiling the techniques and methods used to identify hidden structures within networks.

2.2.1 Communities

The main challenge in graph clustering is finding a clear way to define a community. No definition is broadly accepted. In truth, the definition typically depends on the specific system and/or application one has in mind. The core idea, which guides how we define communities in graphs, is that a community should have more connections between its own members than with the outside of the group[17].

A strong community in a graph, as proposed by Radicchi et al [42], refers to a subset where each vertex has more connections within the subset than outside it, with internal degrees surpassing external degrees.

$$K_{\mathrm{i}}^{\mathrm{in}}(V) > K_{\mathrm{i}}^{\mathrm{out}}(V), \quad \forall i \in V.$$

Alternatively, a weak community is characterized by the subgraph having a higher sum of internal degrees compared to external degrees.

$$\sum_{i \in V} K_{\mathrm{i}}^{\mathrm{in}}(V) > \sum_{i \in V} K_{\mathrm{i}}^{\mathrm{out}}(V).$$

Notably, a strong community also fulfills the criteria for a weak community, though the opposite relationship does not always hold true[42].

Fortunato [17] classifies community definitions into three categories: local definitions, global definitions, and definitions based on vertex similarity. In the context of local definitions, communities are characterized as segments of the graph that exhibit limited connections to the rest of the system. Within this framework, the focus is placed on understanding the internal composition of communities, disregarding interactions with the larger graph.

In contrast, global definitions entail the utilization of an overarching criterion, specific to the graph, to identify communities. Depending on the applied algorithm, this criterion could involve clustering or distance-based metrics. Typically, such criteria demonstrate the presence of a community structure in the graph that diverges from what would be expected in a random graph.

The third approach, based on vertex similarity, interprets communities as collections of vertices that share notable similarities with one another.

2.2.2 Community detection

Community detection is the process of dividing a graph into densely interconnected groups that share similar objects or interactions. This task is crucial for understanding hidden structures within networks and finds diverse applications across fields, ranging from social media to biology and data analysis. This technique has evolved from early research on work groups in government agencies to contemporary algorithms used today[17].

An illustration of this concept can be observed in figure 2.17, where nodes have been grouped into three distinct sets.

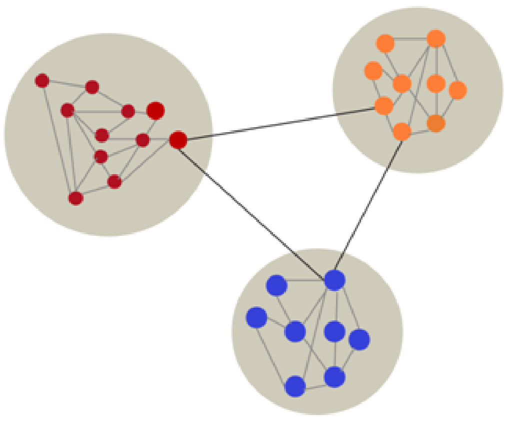


Figure 2.17: Visualization of a Network with 3 Distinct Communities.

2.2.3 Static Community Detection

Static community detection is the technique of discovering groups of nodes in a network that are more densely connected to one another than the rest of the network. There are many algorithms for static community detection, which can be broadly categorized into classical clustering approaches and modularity-based methods. Classical clustering approaches include graph partitioning, which involves dividing vertices into predetermined groups to minimize inter-group edges, and hierarchical clustering, which builds a tree-like structure of nested communities. Modularity-based methods aim to maximize a quality function called modularity, which measures the degree of separation between communities and the density of connections within communities[17].

2.2.4 Dynamic Community Detection

Dynamic community detection is the process of recognizing groups of nodes in a network that change over time [35]. In dynamic networks, communities can evolve through various operations, such as birth, death, merging, splitting, growth, contraction, continue, and resurgence (Figure 2.18)[40]. Dynamic community detection algorithms aim to capture these changes and track the evolution of communities over time. There are two main approaches to dynamic community detection:

- **Instant-optimal approaches** treat each time step as a separate static network and apply static community detection algorithms to each snapshot.
- **Temporal trade-off approaches** consider the past topology and previously identified partitions to define communities at a given time as a compromise between an optimal solution at that time and the known past[45].

Dynamic community detection is a challenging problem due to the complexity of network dynamics and the need to balance accuracy and computational efficiency[45].

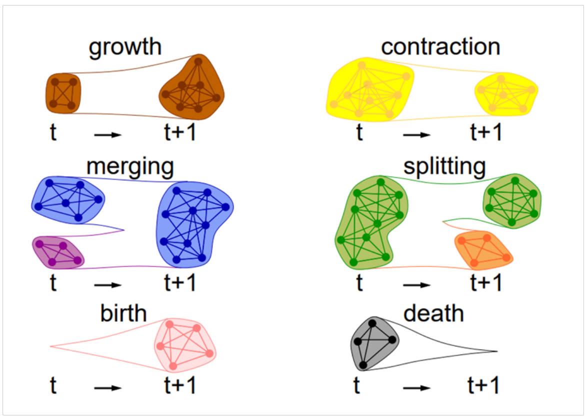


Figure 2.18: Six basic Operations of Community Evolution in Dynamic Networks[17].

2.3 Machine learning

According to Arthur Samuel, machine learning is a subclass of artificial intelligence that allows computers to learn from data without the need for explicit programming.This technology is particularly valuable when dealing with vast datasets that are challenging to interpret manually. Its primary goal is to uncover meaningful patterns in data, facilitating the creation of data-driven models and applications. With the growing availability of datasets, machine learning is increasingly in demand across various industries, offering efficient ways to handle and extract valuable insights from complex information[47].

2.3.1 Machine learning Types

Figure 2.19 illustrates the four primary types of machine learning, each with its own approach and application. These categories are supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

- **Supervised Learning:** Supervised learning is a fundamental notion in machine learning in which models are taught to map inputs to outputs using labelled input-output datasets. This involves learning a function from instances, usually using a dataset partitioned into training and testing subsets. Two important tasks in supervised learning are:Regression and Classification[47].

Regression In regression, the goal is to predict a continuous numerical value, like predicting house prices based on features such as size and location.

Classification: In classification, the aim is to categorize data into distinct classes or categories, like classifying emails as spam or not spam based on their content.

In both cases, supervised learning algorithms rely on external guidance from labeled data to make predictions or classifications, forming the foundation for various practical applications of machine learning.

- **Unsupervised Learning:**

  Unsupervised learning, as opposed to supervised learning, occurs without the presence of correct answers or external guidance. In this realm of machine learning, algorithms autonomously uncover and reveal intriguing patterns and structures within data. These algorithms learn essential features from the data, and when faced with new data, they utilize the previously acquired features to identify patterns or classify data into relevant groups. Unsupervised learning is primarily used for clustering (grouping data points based on similarities) and dimensionality reduction (simplifying complex datasets for efficient analysis by reducing features while preserving vital information, such as Principal Component Analysis or PCA). This approach is particularly valuable for uncovering hidden insights and structures within unlabeled data, without the need for external guidance[47].

- **Semi-supervised Learning:**

  Semi-supervised machine learning combines elements from both supervised and unsupervised techniques, proving highly advantageous in fields like machine learning and data mining, where obtaining labeled data can be challenging and resource-intensive. While traditional supervised learning relies solely on labeled datasets, semi-supervised learning leverages both labeled and unlabeled data, offering a practical solution in scenarios where acquiring labeled data is difficult[47]. Semi-supervised learning finds applications in:

Classification: It improves prediction accuracy by using both labeled and unlabeled data, making it valuable for tasks like medical diagnosis with limited labeled examples.

Clustering: Combines labeled and unlabeled data to uncover meaningful clusters, aiding tasks such as customer segmentation in marketing with partial labels.

- **Reinforcement Learning:**

  Reinforcement learning is a vital branch of machine learning that revolves around the decision-making process of software agents within an environment to maximize cumulative rewards. It stands as one of the core paradigms in machine learning, alongside supervised and unsupervised learning. Unlike the other paradigms, reinforcement learning focuses on teaching agents how to make sequential decisions to achieve optimal outcomes over time, making it particularly relevant for applications where actions have a long-term impact and the environment is dynamic.This learning paradigm uses rewards and penalties to guide decisions, ultimately aiming to maximize rewards and minimize risks[47].

  Positive Rewards: These rewards serve as incentives for agents, reinforcing actions that lead to favorable outcomes, encouraging the agent to repeat those actions in the future.

  Negative Penalties: Penalties in reinforcement learning discourage unfavorable actions by imposing costs or negative consequences, prompting the agent to learn and avoid such actions in subsequent interactions with the environment.

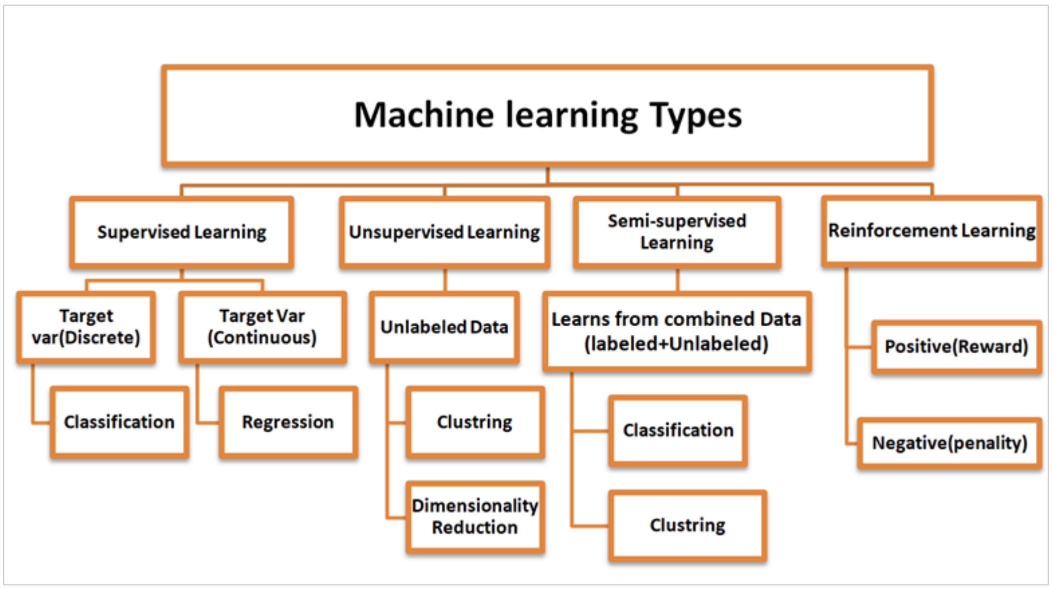


Figure 2.19: Machine Learning Types.

2.3.2 Artificial Neural Networks

An artificial neural network (ANN) is a machine learning technique modelled after biological neural networks (Figure 2.20). Nodes in ANNs communicate via connections (similar to axons and dendrites). Similar to how synapses grow in our brains when neurons have associated outputs, connections in ANNs are weighted based on their contribution to reaching a desired outcome. This notion serves as the cornerstone for how artificial neural networks learn and make decisions.

In a neural network, information is processed through a series of interconnected layers. Each layer consists of individual units called neurons. These neurons take input values (denoted as $x_1, x_2, x_3, \ldots, x_n$) and multiply them by specific weights $(w_1, w_2, w_3, \ldots, w_n)$, which are parameters the network learns during training.

Once the inputs are weighted, the network computes their sum and passes it through an activation function. This activation function helps normalize the sum and produces an output. This output can serve as input for another neuron in a subsequent layer. This capability of neural networks to have multiple layers for information processing is what gives rise to the concept of deep learning[36].

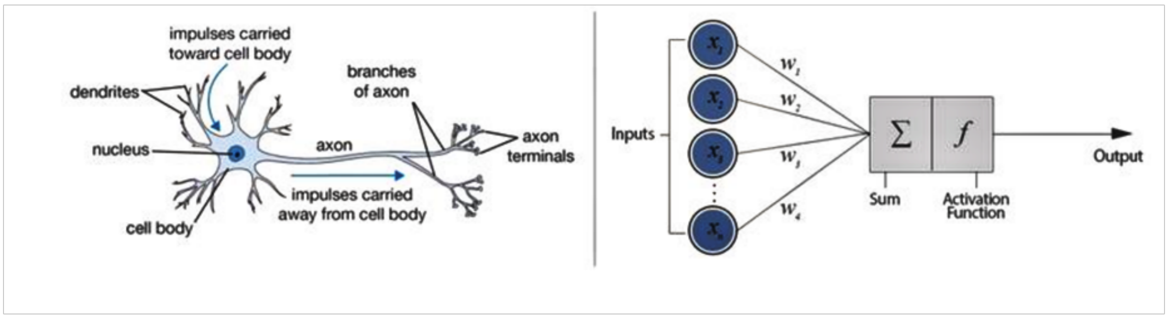


Figure 2.20: Biological neuron vs Artificial neuron [36].

In the following table2.1, we present three commonly used activation functions in neural networks, along with their graphical representations and mathematical formulas.

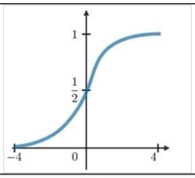
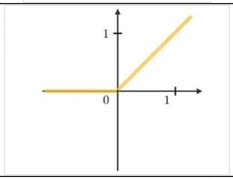
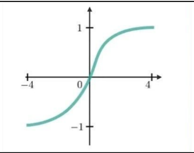
Name of Function	Representation	Formula
Sigmoid		$\sigma(x) = \frac{1}{1+e^{-x}}$
ReLU (Rectified Linear Unit)		$\text{ReLU}(x) = \max(0, x)$
Tanh (Hyperbolic Tangent)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Table 2.1: Activation Functions.

2.4 Graph Neural Networks

In 1997, Sperduti et al. [52] introduced the application of neural networks to directed acyclic graphs, marking a pioneering step in utilizing neural networks for such structures. Later, in 2005, Gori et al.[20] outlined the fundamental concept of Graph Neural Networks (GNNs), which are designed to harness the power of deep neural networks for graph-structured data. This approach represents an extension of convolutions to non-Euclidean data, allowing GNNs to capture intricate relationships. Fundamentally, GNNs are characterized by their reliance on neural message passing, a dynamic process wherein nodes iteratively exchange and refine vector messages through neural networks.

2.4.1 Message Passing Framework

In a Graph Neural Network (GNN), during each iteration of message-passing, we update the hidden representation of each node within the graph.This update is achieved through a combination of aggregation, where information from neighboring nodes is collected, and an update mechanism that integrates this aggregated information with the node's current hidden embedding.

In this section, we elucidate the core stages that delineate the foundational process underpinning the Graph Neural Network (GNN) message-passing framework[25].

At the start of the Graph Neural Network (GNN) message passing process, during iteration 0 ($k = 0$), each node's initial embedding is set to match its input feature. As the process continues through multiple iterations, information is gathered from neighboring nodes. A node's neighbors are the directly linked nodes in the graph, forming its neighborhood $N(u)$. The collected information is combined to create a "message," which carries insights from nearby nodes and captures the local context. This message is then integrated with the node's current hidden embedding ($h_u^{(k)}$) using an "UPDATE" function, often executed by a neural network. This step enables the node to refine its understanding by incorporating knowledge shared by its neighbors. The entire process is repeated for a fixed number of iterations. In each iteration, nodes adapt their embeddings based on information from the previous round, allowing them to gradually improve their embeddings by considering information from farther-reaching nodes in the graph. After completing K iterations, the GNN process concludes, and the resulting embeddings represent the final output of the model.

The equation for the update is as follows:

$$
\begin{aligned}
h_u^{(k+1)} &= \text{UPDATE}^{(k)}(h_u^{(k)}, \text{AGGREGATE}^{(k)}(\{h_v^{(k)}, \forall v \in N(u)\})) \\
h_u^{(k+1)} &= \text{UPDATE}^{(k)}(h_u^{(k)}, m_{N(u)}^{(k)})
\end{aligned}
$$

Where:

$$
\begin{aligned}
h_u^{(k+1)} &: \text{Updated embedding of node } u \text{ after } k+1 \text{ iterations} \\
\text{UPDATE}^{(k)} &: \text{Update function for the } k\text{-th iteration} \\
h_u^{(k)} &: \text{Embedding of node } u \text{ in the } k\text{-th iteration} \\
m_{N(u)}^{(k)} &: \text{Message generated by aggregating information} \\
&\quad \text{from the neighborhood } N(u) \text{ in the } k\text{-th iteration}
\end{aligned}
$$

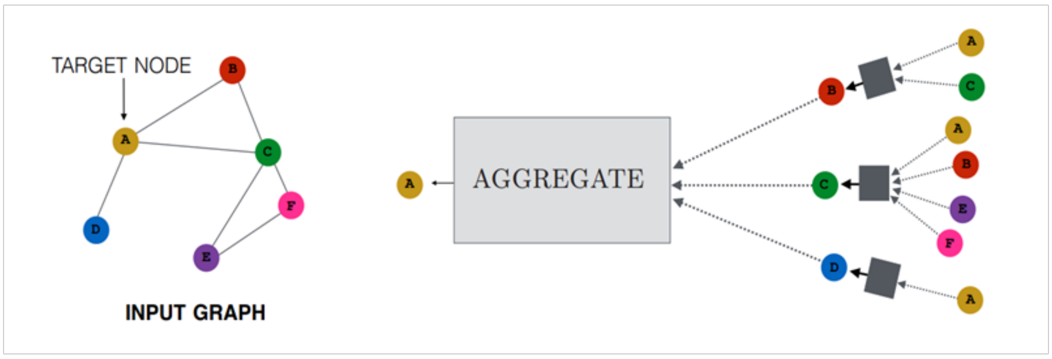


Figure 2.21: Message Aggregation in Two-Layer GNN [25].

Operationalizing the abstract GNN framework involves defining implementable UPDATE and AGGREGATE functions. The fundamental GNN model discussed here, inspired by prior works by Merkwirth and Lengauer [33] and Scarselli et al. [48], presents a basic message passing approach. This approach involves combining messages from neighboring nodes, linearly combining neighborhood information with the current node's embedding, and applying an elementwise non-linearity, as shown in Equation (2.1):

$$h_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neigh}}^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)}\right) \tag{2.1}$$

Where $W_{\text{self}}^{(k)}$ and $W_{\text{neigh}}^{(k)}$ represent trainable parameter matrices, while $h_u^{(k-1)}$ and $h_v^{(k-1)}$ stand for the embeddings of nodes u and v from the previous iteration. Additionally, $b^{(k)}$ corresponds to the bias term employed in the model. The symbol σ represents an elementwise non-linearity function [25].

The aggregation of messages from neighboring nodes is represented by Equation ((2.2)):

$$mN(u) = \sum_{v \in N(u)} h_v() \tag{2.2}$$

The role of the UPDATE function is illustrated by the equation:

$$UPDATE(h_u, mN(u)) = \sigma\left(W_{\text{self}}(h_u) + W_{\text{neigh}}(mN(u))\right)$$

While the aggregation process is expressed by the equation:

$$mN(u) = AGGREGATE(h_v, \forall v \in N(u))$$

This process is similar to a standard multi-layer perceptron (MLP) or an Elman-style recurrent neural network (RNN). The UPDATE and AGGREGATE functions play a key role in this model, enabling the integration of node embeddings and neighborhood information. This foundational GNN framework serves as a starting point for understanding more complex GNN architectures.

2.4.2 Graph Neural Networks Types

In the domain of Graph Neural Networks (GNNs), three fundamental types emerge as noteworthy: Graph Convolutional Networks (GCNs), GraphSAGE, and Graph Attention Networks (GATs).

Graph Convolutional Networks (GCNs)

A Graph Convolutional Network (GCN), a type of Graph Neural Network (GNN) introduced by Kipf and Welling [29] extends convolutional operations to graphs. This innovation enables the network to effectively learn from the neighborhoods of nodes within a graph, much like how convolutional layers operate in convolutional neural networks (CNNs). GCNs are widely employed in tasks such as node classification, link prediction, and recommendation. The central propagation rule of a GCN, The propagation rule can be expressed as:

$$H^{(k+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)}W^{(k)}\right) \tag{2.3}$$

Where $\tilde{A}$ represents the adjacency matrix of the graph, and $\tilde{D}$ represents its degree matrix, both of which include self-loops. $H^{(k)}$ denotes the activations in the k-th layer, where $H^{(0)}$ represents the input data X. $W^{(k)}$ stands for the weights associated with the k-th layer. This rule captures how information is gathered and transformed across layers in the Graph Convolutional Network (GCN).

Graph Attention Networks

Graph Attention Networks (GAT) are a significant advancement in the field of communication networks and various other domains. GATs are distinguished by their incorporation of attention mechanisms, which enable them to prioritize connections within a network. In a GAT layer, node information and features undergo a learnable linear transformation using a weight matrix. Subsequently, self-attention mechanisms determine the relevance of neighboring nodes. The attention coefficients are calculated as follows:

$$e_{ij} = a(W\hat{h}_i, W\hat{h}_j)$$

and then normalized using softmax:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{e^{e_{ij}}}{\sum_{k \in N_i} e^{e_{ik}}}$$

Here, ”a” represents the attention mechanism, a neural network with LeakyReLU activation. These equations guide the aggregation of features from neighboring nodes, facilitating the extraction of meaningful information from complex networks[53].

GraphSAGE (Graph Sample and Aggregated)

The Graph Sample and aggregate (GraphSAGE) is a distinctive graph neural network (GNN) algorithm known for its efficient handling of large graphs through random sampling and was developed by Hamilton et al. [24]. Unlike many other GNNs, GraphSAGE can manage both space and time complexity independently of graph properties like node degree distribution and batch size. Its core features include using fixed-size random subsets for sampling, multiple graph convolutional layers (K layers) for aggregating information, and differentiable aggregator functions (AGG_k) for each layer (k) to combine data from neighboring nodes [11].

2.4.3 Graph Neural Networks Applications

Indeed, Graph Neural Networks (GNNs) play a crucial role in various domains[23]:

- **Social Networks:** GNNs personalize content and strengthen friend connections.
- **Drug Interaction Prediction:** They identify potential side effects resulting

from drug interactions.

- **Traffic Prediction:** GNNs accurately forecast traffic patterns for efficient transportation systems.
- **E-Commerce Recommendations:** GNNs enhance product recommendations for online shoppers.
- **Text Analysis:** They structure textual data, improving analysis and understanding.
- **Molecular Structure Analysis:** GNNs excel in analyzing molecular structures represented as graphs[23] .

2.5 Autoencoder Technology

The concept of autoencoders was first introduced in a seminal paper by Hinton and Salakhutdinov [26] as an artificial neural network designed to capture feature representations. It consists of two fundamental components: an encoder and a decoder (Figure 2.22). The encoder transforms input data into a lower-dimensional representation to effectively capture essential features. Subsequently, the decoder reverses this process, with the goal of reconstructing the original input as accurately as possible. The primary objective of autoencoder training is to autonomously learn a data representation that proves valuable for various applications, including dimensionality reduction, clustering, anomaly detection, and feature representation.

2.5.1 Different Autoencoder architecture

Autoencoders exhibit diverse variations, with three commonly employed types including Denoising Autoencoders, Variational Autoencoders for probabilistic modeling, and Convolutional Autoencoders designed for image-centric tasks.

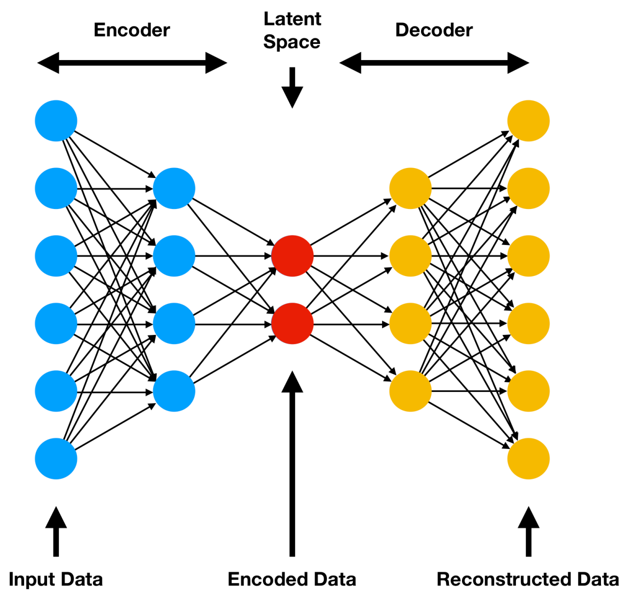


Figure 2.22: Autoencoder Architectur.

Convolutional Autoencoders (CAEs)

Convolutional Autoencoders (CAEs) are neural networks designed for image tasks, using convolutional layers to extract features while preserving spatial info. They create compact data representations by constraining the embedded layer's dimension, avoiding direct input-output copying. In CAEs, encoding uses convolution and activation (e.g., ReLU) to produce a concise data representation, as shown in Equation (2.4). While decoding uses transpose convolution, the decoder is defined by equation (2.5). CAEs minimize Mean Squared Error (MSE) to capture and reproduce essential image features, making them valuable for feature extraction and image data compression [22].

$$h = \sigma(x * W) \tag{2.4}$$

$$x_0 = \sigma(h * U) \tag{2.5}$$

Figure 2.23 below illustrates the Convolutional Autoencoder (CAE) architecture for MNIST.

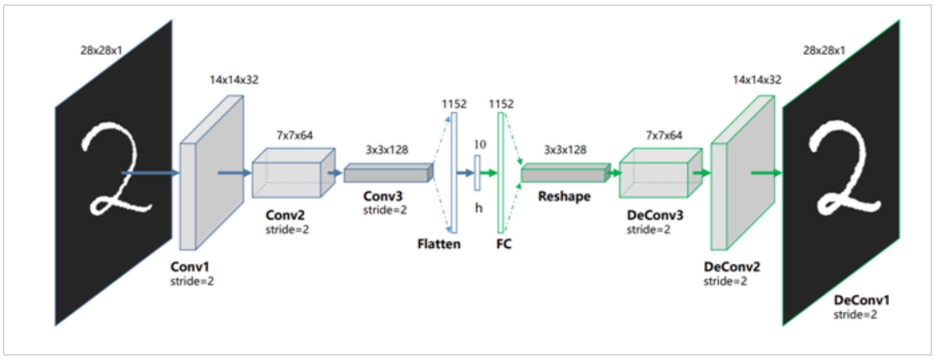


Figure 2.23: Convolutional Autoencoder Architecture [22].

Denoising Autoencoders (DAE)

A Denoising Autoencoder (DAE) comprises three layers: an input layer, a hidden layer (also called an encoding layer), and an output layer (decoding layer). It operates by first transforming input data, x, into a vector x by introducing noise or setting some elements to zero. The encoding phase utilizes a nonlinear transformation:

$$y = f_e(Wx + b)$$

Here, y represents the hidden layer output (feature representation), W are the input-to-hidden weights, b is the bias, and $f_e()$ is the activation function, often using ReLU to create sparse features.

For decoding or reconstruction, DAE uses a mapping function $f_d()$:

$$z = f_d(W^T y + b^d)$$

Where z is the DAE's output, effectively the reconstruction of the original data x. The output layer mirrors the input layer's dimensions, and W^T refers to tied weights. Depending on the input range, the decoding function $f_d()$ can be softplus or linear. DAE is trained to minimize a reconstruction-oriented cost function, which varies based on the input range. It includes the cross-entropy function for input values in [0, 1], and the square error function otherwise, with an added regularization term (W_2) controlled by the parameter λ. Training employs minibatch stochastic gradient descent (MSGD) for optimization [12].

Variational AutoEncoders (VAEs)

Variational autoencoders (VAEs) are widely used for developing deep generative models. They are particularly valuable for tasks such as data compression and generating new content, making them versatile tools in various fields. VAEs work by learning to encode input data into a lower-dimensional latent space and then decoding it back to the original data space. In this latent space, VAEs model a probabilistic distribution characterized by mean and standard deviation outputs, enabling them to capture uncertainty and generate diverse and meaningful samples (see figure 2.24).

In VAEs, the encoding step transforms input data into a latent vector z, denoted as $\mathbf{z} = \mathbf{Encoder}(\mathbf{x}) \sim \mathbf{q}(\mathbf{z}|\mathbf{x})$, where x is the input data, and $q(z|x)$ represents the approximate posterior distribution.

The decoder step reconstructs the latent vector z back into an image, producing an output $\bar{x}$, denoted as $\bar{\mathbf{x}} = \mathbf{Decoder}(\mathbf{z}) \sim \mathbf{p}(\mathbf{x}|\mathbf{z})$, which follows the generative distribution $p(x|z)$ [28].

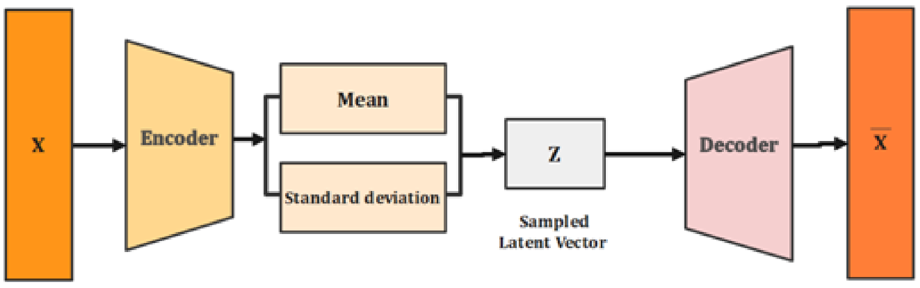


Figure 2.24: Variational Autoencoder Architecture.

Sparse Autoencoder(SAE)

A sparse autoencoder is a type of neural network that is used for unsupervised learning. It learns to compress the input data into a lower-dimensional representation by training the network to reconstruct the original data from the compressed form. The sparse autoencoder distinguishes itself by imposing a sparsity constraint on the hidden units, allowing only a small percentage of them to be active at any given time. This motivates the network to develop a compact and efficient representation of the input data. The sparsity constraint is achieved by adding a penalty term to the objective function that encourages the hidden units to be activated only for a small fraction of the input data. The sparse autoencoder has been shown to be effective in learning features from unlabeled data, which can be useful in a variety of applications such as computer vision, speech recognition, and natural language processing[39].

2.5.2 Autoencoders Applications

Autoencoders have several practical applications, including:

1. **Dimensionality Reduction:** Autoencoders are used to reduce the dimensionality of data efficiently. Unlike traditional methods like PCA, autoencoders can handle large datasets with mini-batch training, making them suitable for big data applications. Autoencoders also allow for non-linear transformations of features, offering greater flexibility.

2. **Classification with Latent Features:** Autoencoders can be employed for classification tasks using the latent features they learn. By training a classifier on these features, you can achieve significant reductions in training time while maintaining reasonable accuracy, especially in high-dimensional datasets.

3. **Anomaly Detection:** Autoencoders are effective for anomaly detection. By training on a dataset without anomalies and then measuring the reconstruction

error of each data point, you can identify outliers or anomalies in the dataset. This is valuable for fraud detection, fault detection, and quality control.

4. **Denoising Autoencoders:** Denoising autoencoders are trained to remove noise from input data. They can be used to clean noisy images, reconstruct corrupted text, or restore data affected by various types of noise. Denoising autoencoders learn to generate clean outputs from noisy inputs[34].

2.6 Conclusion

In conclusion, this chapter explores fundamental concepts in graph theory, community detection, graph neural networks (GNNs), and autoencoders. These concepts provide essential frameworks for understanding network structures, identifying communities within graphs, and utilizing advanced techniques like GNNs and autoencoders for data analysis and representation learning.

In the next chapter, we examine in detail the state of the art of community detection methods.

Chapter 3

State Of The Art

With the increasing need to analyze networks, many different community structure detection approaches have been proposed. The exploration of community detection methods has evolved over the years, encompassing a spectrum of approaches ranging from traditional methodologies to deep learning techniques. In this chapter, we embark on a comprehensive exploration through the state of the art in community detection, delineating the landscape into two distinct realms: traditional methods and deep learning methods. These methods are summarized in the Figure 3.1.

3.1 Traditional Methods

In this section, we explore traditional approaches for community detection, focusing on hierarchical clustering, modularity optimization, and dynamical community detection methods.

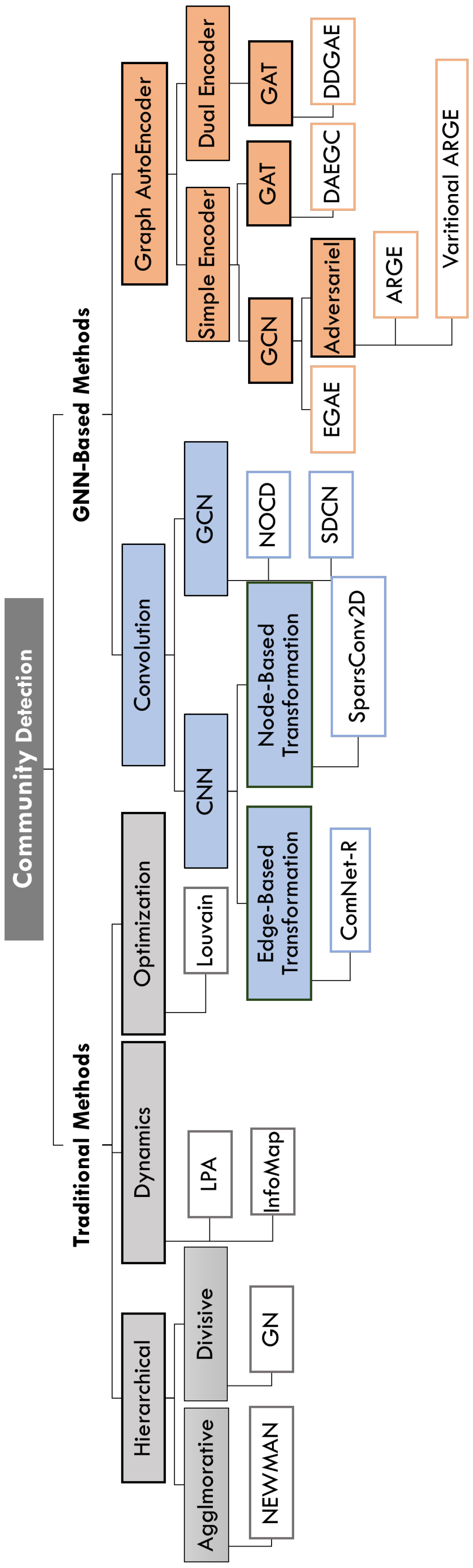


Figure 3.1: Traditional and deep learning community detection methods.

3.1.1 Hierarchical Clustering Methods

Hierarchical clustering is a method of community analysis that seeks to build a hierarchy of communities. When applied to networks, hierarchical clustering involves grouping nodes based on their connectivity or similarity. This helps in identifying communities within the network. This category of methods can be represented by a dendrogram, a tree-like diagram that records the sequences of merges or splits in hierarchical clustering (see Figure 3.2). There are two main types of hierarchical clustering:

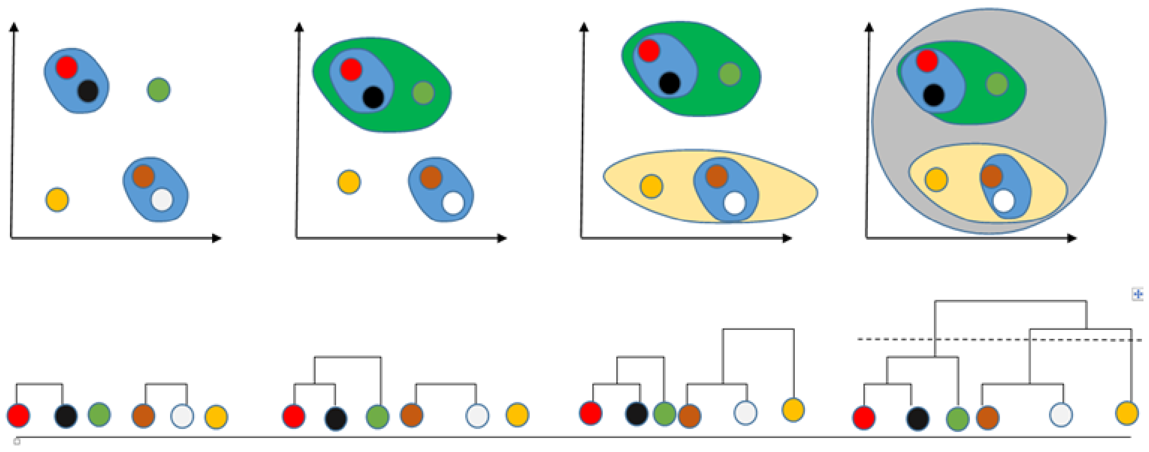


Figure 3.2: Hierarchical Clustering Steps represented by Dendrogram.

3.1.1.1 Agglomerative Algorithms

Agglomerative algorithms employ a bottom-up technique, initially treating each node as an independent cluster and iteratively merging them based on high similarity[17]. *Newman's Greedy Search Algorithm* exemplifies this approach, proposed by Mark Newman in 2004. In the initial stages, each node is designated as a separate community, forming n initial communities. The algorithm 4 then iteratively calculates distances between communities and merges the closest pairs, guided by modularity gain. This process continues until the entire graph converges into a cohesive community.

Notably, Newman's algorithm demonstrates performance with a time complexity

Algorithm 4 Newman's Algorithm

```
procedure NewmanAlgorithm($G$)
    Initialize each node as its own community
    Calculate the modularity of the initial partition
    while there are communities to merge do
        for each pair of mergeable communities do
            Calculate the modularity of the merged communities
        end for
        Merge the two communities that maximize modularity gain
        Update the list of merged communities
    end while
end procedure
```

of $O(n^3)$ [38].

3.1.1.2 Divisive Algorithms

Divisive algorithms use a top-down approach, first treating the entire network as a single community and then iteratively separating it by removing links between nodes with low similarity, resulting in distinct communities. The divisive approach in community detection begins by placing all nodes into a single group. It then aims to break down the network into multiple communities while progressively removing edges that link nodes with low similarity.As a result, the entire graph divides into several smaller segments. This iterative process continues until communities consist of individual nodes [17].

The Girvan-Newman algorithm, introduced by Michelle Girvan and Mark Newman in 2002, stands out as a prominent divisive approach within community detection. This innovative method (algorithm 5) focuses on gradually removing edges that serve as important connectors for the maximum number of shortest pathways between nodes. The algorithm's essence is the systematic calculation of betweenness centrality for each edge in the graph. It entails eliminating the edge with the highest betweenness centrality, recalculating betweenness centrality for the remaining edges, and continuing this process until no edges remain.

The betweenness centrality of a node $BC(v)$ is the fraction of shortest paths between all pairs of nodes in the network that pass through that node. Nodes with high betweenness centrality are frequently seen as critical to maintaining network connectivity and efficient information flow. The formula for betweenness centrality for a node v is commonly represented as:

$$BC(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where σ_{st} represents the total number of shortest paths from node s to node t, while $\sigma_{st}(v)$ is the number of shortest paths from s to t that pass via node v. The summing is performed on all possible pairs of nodes s and t (excluding v).

Algorithm 5 Girvan-Newman Algorithm

```
procedure GirvanNewman(G)
    while $|E| > 0$ do
        Calculate the betweenness centrality of all edges in $G$
        Find the edge with the highest betweenness centrality
        Remove the identified edge
        Identify communities based on the connected components of $G$
    end while
end procedure
```

The Girvan-Newman algorithm's time complexity is $O(m^2 n)$, where m is the number of edges and n is the number of vertices. This complexity arises from the repeated calculation of betweenness centrality for each of the m edges, with each calculation taking $O(mn)$ time[18].

3.1.2 Modularity Optimization Methods

The concept of modularity, initially introduced by Girvan and Newman, is employed by multiple algorithms.Initially, a stopping criterion for the Girvan-Newman algorithm was suggested. [17]. Modularity can then be written as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Where m is the weighted total of edges in the community network, and c_i is the community where the node i resides. A_{ij} is the element of the adjacency matrix between nodes i and j, k_i is the sum of edge weights associated to node i, and the function $\delta(C_i, C_j)$ shows whether nodes i and j belong to the same community.

$$\delta(C_i, C_j) = \begin{cases} 1 & \text{if } C_i = C_j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

To discover communities, modularity optimization algorithms find out the community division with the highest modularity among all possible community divisions in the network. The first algorithm developed to enhance modularity was *Newman's greedy method.* It is an iterative agglomerative hierarchical clustering method that progressively combines vertex groups to form larger communities, aiming to increase modularity (refer to 3.1.1.1).

Blondel et al. introduced a greedy algorithm for weighted graphs, commonly known as the *Louvain Algorithm* [4]. Initially, each vertex is assigned to its own community. The algorithm begins with a sequential pass over all vertices. For a given vertex i, it calculates the gain in weighted modularity from moving i to the community of its neighbor j, selecting the neighbor's community that offers the highest positive increase in modularity. This forms the initial partition.

In the next step, communities are replaced by supervertices, which are connected if the corresponding communities share edges. The edges between supervertices have weights equal to the sum of the weights of the edges between the lower-level communities. The algorithm iterates these steps, creating new hierarchical levels and supergraphs, and recalculating modularity changes after merging or splitting vertex groups. The process stops when modularity can no longer increase [5]. Figure 3.3 depicts the Louvain algorithm, whereas Algorithm 6 provides a pseudocode summary. The time complexity is O(m), where m is the number of edges in the graph.

Algorithm 6 Louvain Algorithm

```
procedure LOUVAIN($G$)
    Input: Network graph $G(V, E)$
    Output: Partition of nodes $V$ into communities
    Initialization: Assign each node to its own community
    while there is a node $v$ in $G$ do
        Optimize Modularity:
        Move $v$ to community that maximizes modularity
    end while
    while more than two communities exist do
        Aggregate Communities:
        Merge communities with the highest modularity gain when merged
    end while
    Output: Resulting partition of nodes into communities
end procedure
```

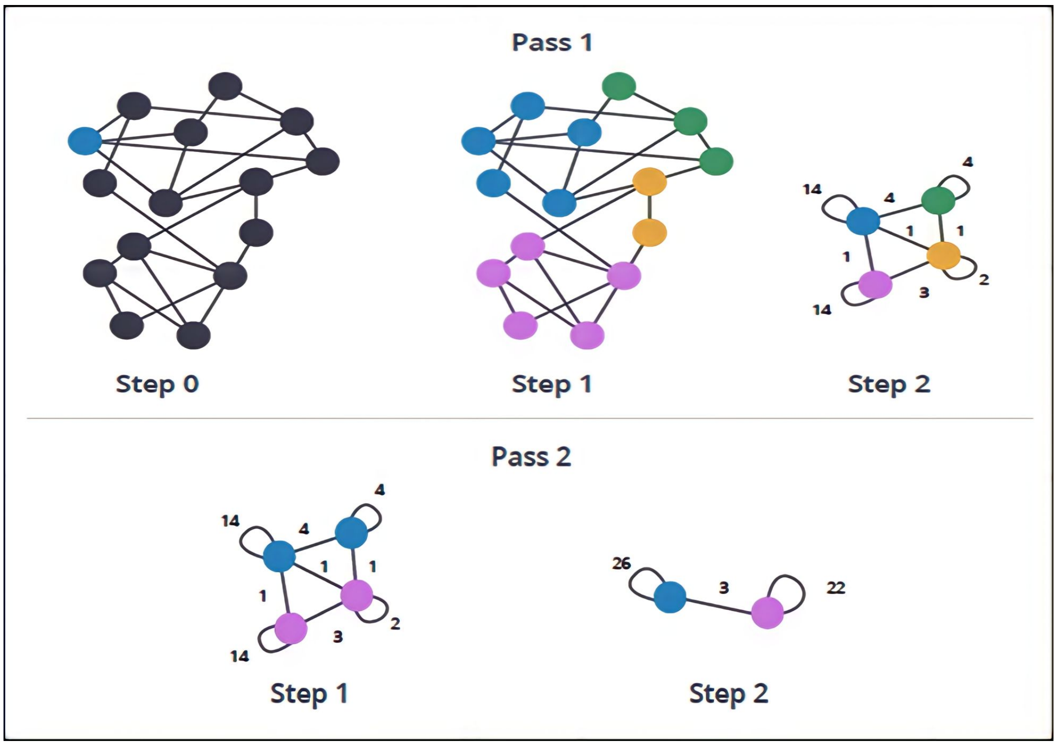


Figure 3.3: Louvain Algorithm Steps.

3.1.3 Dynamical Methods

Dynamic approaches for community detection refer to the process of understanding different groups of nodes in a network over time. Among the various methods, we're focusing on the InfoMap and Label Propagation algorithms

3.1.3.1 Label Propagation Algorithm (LPA)

Raghavan, Albert et al. [44] developed a label propagation algorithm (LPA) which propagates node labels throughout the network. Initially, each node is assigned a distinct label, which the algorithm iteratively changes in a random sequential order. During each cycle, a node adjusts its label based on the label shared by the greatest number of its neighbors. If a node has numerous candidate labels to update, one is chosen at random. The label update rule for a node u in an undirected and unweighted network with n nodes is expressed as:

$$l_{\text{new}_u} = \arg\max_{l'_u} \sum_{v=1}^{n} A_{uv}\delta(l'_u, l_v),$$

Where l_{new_u} is the new label assigned to node u, l'_u represents a label of the neighbors of u, v is a node in the network, l_v is the label of node v, A_{uv} is the element of the adjacency matrix indicating the connection between node u and node v, and δ is the Kronecker delta function (1 if l_v and l'_u are the same, 0 otherwise).

The label updating method continues until node labels remain unchanged after an iteration . Communities are therefore defined as groupings of nodes with the same label. To ensure algorithm convergence, node labels are changed asynchronously. This means that each node's label changes dependent on the labels of certain neighbors from the previous iteration and the labels of other neighbors from the current iteration.

The pseudocode of the Label Propagation Algorithm (LPA) is outlined in Algorithm 7. LPA has a nearly linear temporal complexity of $O(m)$, where m is the number of network edges[44].

Noting a drawback of the Label Propagation Algorithm (LPA), it tends to assign the same label to all nodes, potentially impeding a meaningful interpretation of the detected community structure[2].

Algorithm 7 Label Propagation Algorithm (LPA)

```
procedure LabelPropagation(Graph $G$)
    Initialize each node with a unique label
    Set maximum number of iterations: $max\_iterations$
    $iterations \leftarrow 0$
    while $iterations < max\_iterations$ do
        for each node $u$ with label $l$ in a sequential order of nodes do
            $l'$ = the neighboring label of $u$ shared by the maximum number of neighbors of $u$
            if $l' \neq l$ then
                Assign label $l'$ to node $u$
            end if
        end for
        if labels of every node remain unchanged after an iteration then
            return the current node label assignment
        end if
        $iterations \leftarrow iterations + 1$
    end while
end procedure
```

3.1.3.2 InfoMap algorithm

The InfoMap algorithm, introduced by Rosvall and Bergstroin in 2008 [46] , offers a unique perspective on community detection within networks. Rather than directly identifying communities, it approaches the problem by minimizing the code length of a random walk through the network. The underlying idea is to represent each node with a binary Huffman coding, allowing paths in the network to be expressed as specific codewords. This representation is then used to compress the information coding, turning the community detection problem into one of coding compression[46].

The algorithm aims to reduce the description length of random walk paths in the network, utilizing the map equation[46]. This equation is instrumental in dividing communities and finding a compact representation by minimizing the description length of random walk paths. It operates on the concept of representing paths with two levels: the first level names communities, and the second level names nodes within communities using Huffman coding. This ingenious idea allows for the reuse of internal node encodings across different communities, ultimately shortening the length of the encoding.

The map equation calculates the minimum description length for the random walk path, as expressed in Equation 3.2:

$$L(M) = q_{\curvearrowright} H(Q) + \sum_{i=1}^{m} p_i \curvearrowright H(P_i) \tag{3.2}$$

This equation comprises two terms: the first represents the entropy associated with transitions between modules, and the second denotes the entropy within modules, accounting for module exits as movements. Each term is weighted by its frequency in the specific partitioning. Here, q represents the likelihood of the random walk transitioning between modules at any given step. $H(Q)$ denotes the entropy related to module names, specifically the entropy of the codewords. Similarly, $H(P_i)$ represents the entropy related to within-module movements, including the exit code for the module i. The coefficient p_i signifies the proportion of within-module movements in module i, along with the probability of exiting the module i. It is crucial to note that $1 \leq i \leq m$ and $\sum_{i=1}^{m} p_i = 1 - q$.

Algorithm 8 InfoMap Algorithm

```
Require: Graph $G = (V, E)$,minimum quality improvement threshold $\tau$.
Ensure: Node-to-module map $M$.
  Run PageRank to calculate the visit rates for each vertex: $M = \{m_i = v_i \mid v_i \in V\}$.
  $L = L(M)$
  repeat
      $L_{\text{prev}} = L$
      $R$ = random sequence of integers from 1 to $N$
      for $i = 0$ to $N - 1$ do
          $m_{\text{new}} = \text{bestNewModule}(M, v_{R[i]})$
          Move $v_{R[i]}$ to module $m_{\text{new}}$ and update $M$ and $L$
      end for
  until $L_{\text{prev}} - L < \tau$
  return $M$
```

The computational complexity of the community discovery process with the InfoMap algorithm is $O(N \log N)$, where, N is the number of nodes in the network. This complexity is contingent on the network structure, as optimizing the map equation involves traversing the network multiple times.

3.1.4 Comparison between Traditional Methods

In Table 3.1, an overview of a summary of traditional algorithms for the community detection task is presented, including hierarchical, optimization-based, and dynamic approaches. Hierarchical clustering (Newman-Girvan) and GN offers detailed hierarchies but is computationally demanding. Optimization-based clustering (Louvain) is efficient and scalable with high modularity but can vary in results. Dynamic clustering (LPA) is extremely fast but can produce unstable results, and InfoMap balances efficiency and stability with a robust information-theoretic approach but can be computationally intensive. However, the increasing complexity and scale of modern networks motivate the use of deep learning techniques, which can automatically learn and adapt to intricate patterns, offering enhanced scalability and potentially more accurate community detection.

Table 3.1: Summary of Traditional Algorithms for Community Detection

Category	**Algorithm**	**Approach**	**Key Features**	**Advantages**	**Disadvantages**
Hierarchical Clustering:Divisive	Newman [38]	Modularity-based community merging	Modularity maximization, iterative merging	Clear community structures, no need for initial community number	Computationally intensive for large networks
Hierarchical Clustering: Agglomerative	Newman-Girvan (NG) [19]	Divisive, uses edge betweenness to reveal community structure	Edge betweenness, dendrogram construction, O((m+n)n) complexity	Clear hierarchy, no need for prior community number	Computationally expensive, sensitive to edge removal order
Optimization-Based Clustering	Louvain [4]	Modularity optimization, merges communities iteratively	Two-phase process, modularity optimization, O(n log n) complexity	High computational efficiency, scalable, high modularity partitions	Varying results, struggles with small/dense communities
Dynamic Clustering	LPA [44]	Propagates labels based on majority voting among neighbors	Label dynamics, linear complexity O(m)	Extremely fast, no need for prior community number	Unstable solutions, large community production
Dynamic Clustering	InfoMap [46]	Uses random walker to minimize description length of walk	Random walk dynamics, efficient for large networks	Detects hierarchical structures, robust information-theoretic basis	Sensitive to walk parameters, computationally demanding

3.2 Deep Learning Methods

The landscape of community detection has undergone a significant transformation, shifting from traditional shallow methods to the realm of deep learning approaches. In this section, our focus centers on providing an overview of Convolutional Network-Based Community Detection, with a particular spotlight on the integration of Autoencoder-based methods.

3.2.1 Convolutional Based Community Detection

Community detection using convolutional network models encompasses both Convolutional Neural Networks (CNNs) and Graph Convolutional Networks (GCNs). We provide a comprehensive overview of significant contributions from researchers, highlighting advancements in leveraging CNNs and GCNs for efficient community detection across a variety of network structures.

3.2.1.1 CNN-based Community Detection

Xin et al.[56] introduced an innovative Convolutional Neural Network (CNN) approach tailored for community detection in networks with incomplete structural information, such as social networks with gaps in individual data. Operating on 1D matrices representing adjacency relations, the CNN model incorporates convolutional layers for local feature extraction, max-pooling for map generation, and a fully connected layer for community classification. Experimental results on datasets like Football, LiveJournal, and YouTube showcase the model's superiority over both unsupervised and supervised methods, particularly in networks with substantial missing data. While acknowledging achievements, the study emphasizes the need for further exploration of preprocessing and deep learning techniques for ongoing improvements.

Cai et al. [9] made notable contributions with the ComNet-R algorithm, introducing an Edge-to-Image (E2I) model that converts edge structures into images. Their method involves constructing a Community Network (ComNet) to classify edges within the same community or between different communities. The approach streamlines local community views through a breadth-first search based on edge classification and optimizes community structures by merging preliminary communities using the local modularity measure R. Experimental results underscore the effectiveness of their deep convolutional neural network-based edge classification in diverse network scenarios, though the current implementation is limited to undirected graphs.

Turning to A.Bekkair et al.[3] exploration, they address the crucial task of community detection in complex networks using Convolutional Neural Networks (CNNs). Their approach is categorized into node-based and edge-based transformations, with a comparative analysis revealing the node-based method's straightforward and computationally efficient nature, achieving superior results across diverse datasets. However, the edge-based approach, while promising, introduces complexities with multi-step processes, including image transformation and modularity-based algorithms, leading to resource-intensive implementations. The study contributes valuable insights into community detection methodologies while underscoring the computational challenges associated with the edge-based strategy.

3.2.1.2 GCN-based Community Detection

Graph Neural Networks, and specifically Graph Convolutional Networks (GCN), have emerged as powerful tools for community detection tasks. Overlapping community detection in networks poses a critical challenge in machine learning, necessitating the identification of densely connected node groups. The Neural Overlapping Community Detection (NOCD) model, recently introduced by Shchur and Günnemann[50], addresses this by integrating a Bernoulli-Poisson probabilistic model with a two-layer graph convolutional network (GCN).This integration facilitates the learning of community affiliation vectors by minimizing the negative log-likelihood of the Bernoulli-Poisson model.The model's efficacy was demonstrated across diverse real-world datasets, showcasing superior accuracy and efficiency compared to existing methods, particularly in social and citation networks.The model incorporates a thresholding technique to enhance community identification by removing weak affiliations. The results presented underscore the viability of deep learning for graphs as a robust framework for overcoming challenges in overlapping community detection, suggesting a need for increased attention to this approach in future research.

Deyu Bo et al.[6] pioneered the Structural Deep Clustering Network (SDCN) to revolutionize deep clustering by seamlessly integrating structural information. Focused on enhancing the effectiveness of data representations for clustering within a deep learning context, the authors introduced SDCN, leveraging a delivery operator to transfer autoencoder-learned representations to a Graph Convolutional Network (GCN) layer. A dual self-supervised mechanism unifies diverse neural architectures, facilitating comprehensive model updates. Demonstrating consistent superiority over existing techniques, SDCN exhibits enhanced clustering performance and computational efficiency across diverse datasets. However, the model's applicability in large-scale datasets or real-time scenarios may be hindered by its computational complexity, while the quality of input data intricately influences clustering outcomes. These nuances present intriguing aspects for further exploration in the evolving landscape of deep clustering research.

3.2.2 Autoencoder-Based Community Detection

Recent advances in deep learning, particularly autoencoder-based methods, have significantly enhanced community detection within networks. In this section, we introduce a taxonomy of Graph Autoencoders (GAEs), categorizing them into GAE-based simple encoders and GAE-based dual encoders. Within each category, further classification is based on the type of architecture used (see Figure 3.1).

3.2.2.1 Embedding Graph Auto-Encoder

In the realm of graph clustering, the Embedding Graph Auto-Encoder (EGAE) model [58] emerges as a pioneering solution, blending the capabilities of graph autoencoders (GAEs) with clustering techniques to partition nodes effectively.The architecture of EGAE plays a pivotal role in its success, designed with a specific structure that harnesses the strengths of GAEs and relaxed k-means clustering in a synergistic manner.

At the heart of EGAE's architecture lies the encoder-decoder framework, comprising an encoder and dual decoders (as showen in Figure 3.4). The encoder's task is to map input graph data and features into a latent feature space using graph convolution layers, with inner-products serving as a metric. This utilization of inner-products leverages the orthogonal property of representations generated by GAEs, enhancing the model's ability to capture essential graph structures effectively.

The dual decoders in EGAE are instrumental in the clustering process, with one decoder dedicated to reconstructing the graph based on inner-products and the other focused on clustering using relaxed k-means. By simultaneously learning representations and performing clustering, EGAE ensures that the deep features generated by the neural network are well-suited for the specific clustering model.

Through extensive experimentation on benchmark datasets such as Cora, CiteSeer, and Wiki, EGAE demonstrates remarkable performance improvements over traditional clustering methods and state-of-the-art graph embedding techniques. Significant enhancements in clustering accuracy, normalized mutual information, and adjusted rand index across all datasets underscore the model's effectiveness in accurately capturing latent structures and partitioning graphs.

However, EGAE is not without its limitations. Sensitivity to hyperparameters, particularly the tradeoff coefficient α, poses challenges, and scalability concerns may arise with larger datasets. Further exploration and refinement of EGAE's applicability in real-world scenarios are warranted to address these limitations.

H

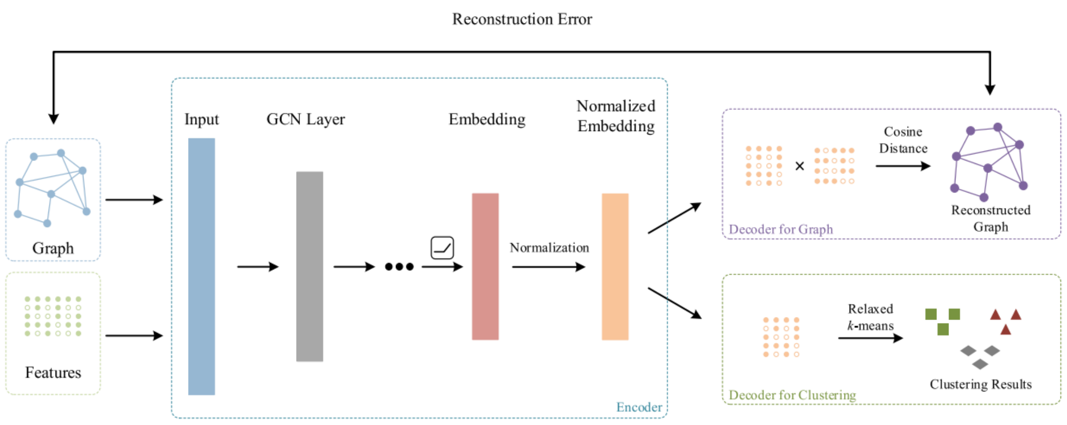


Figure 3.4: Architecture of Embedding Graph Auto-Encoder [58].

3.2.2.2 Adversarially Regularized Graph Autoencoder

The proposed adversarially regularized graph embedding framework, as presented in [41], addresses the challenge of learning robust representations of graph data by considering both the topological structure and node content. This framework integrates adversarial regularization techniques within a graph autoencoder architecture to produce embeddings that effectively preserve the graph's inherent structure while capturing crucial features of the nodes. ARGE (Adversarially Regularized Graph Embedding) and ARVGE (Adversarially Regularized Variational Graph Embedding) are two innovative models developed within this framework. Their architecture comprises two key components: the graph autoencoder and the adversarial network (Figure 3.5). The graph autoencoder utilizes graph convolutional networks (GCNs) to encode the input graph's structure and node content features into low-dimensional embeddings. This involves an encoder model that applies GCN layers to process the adjacency matrix and node features, generating latent representations of the input graph. Subsequently, a decoder model reconstructs the graph structure based on these learned embeddings, preserving essential topological and content-related information. The adversarial network introduces adversarial training to regularize the learned embeddings, ensuring they match a specified prior distribution. It includes a discriminator model trained to distinguish between embeddings generated by the graph autoencoder and samples

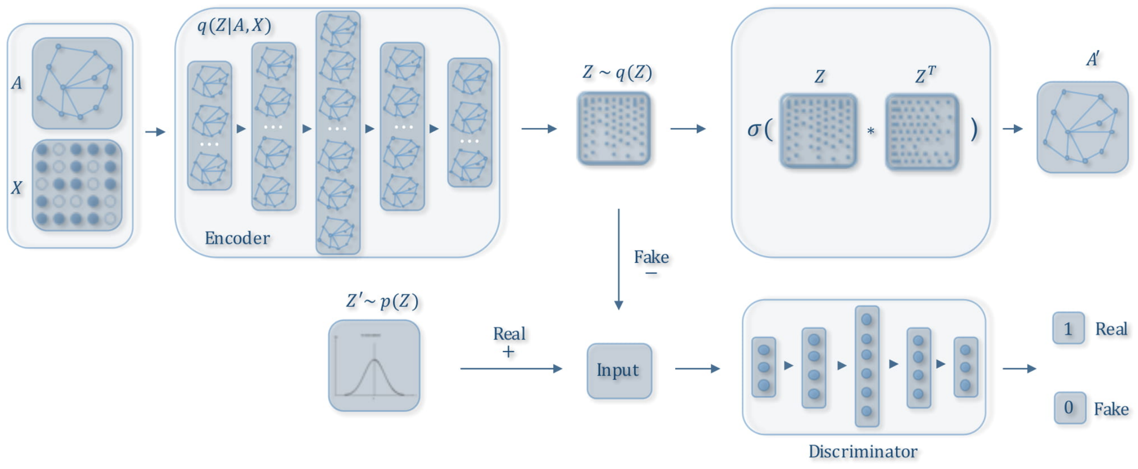


Figure 3.5: Architecture of Adversarially Regularized Graph Autoencoder [41].

drawn from the prior distribution. By minimizing the cross-entropy loss between the discriminator's predictions and ground truth labels, the adversarial network guides the graph autoencoder to produce embeddings that are more stable and consistent with the prior distribution. This combined architecture enables ARGE and ARVGE to learn robust representations of graph data, making them effective for various graph analytics tasks.

In the experiments conducted on real-world graph datasets such as Cora, CiteSeer, and PubMed, ARGE and ARVGE consistently outperform traditional methods and more recent graph embedding approaches across various tasks, including link prediction, node clustering, and graph visualization. The adversarial regularization incorporated into these models contributes significantly to their superior performance, enabling them to achieve high accuracy, AUC, AP, normalized mutual information, F1 score, precision, and average rand index scores compared to baseline methods.

However, despite the advancements, there are limitations to consider. The computational complexity of some techniques, the need for large amounts of training data, and the challenge of generalizing to different graph structures pose constraints on the scalability and applicability of these methods. Additionally, the interpretability of the learned embeddings and the robustness to noisy or incomplete data remain areas for further exploration and improvement in the field of graph embedding.

3.2.2.3 Deep Attentional Embedded Graph Clustering

The Deep Attentional Embedded Graph Clustering (DAEGC)[54] algorithm is a novel approach designed to address the challenges in graph clustering by focusing on attributed graphs. DAEGC stands out for its ability to effectively capture both the structural relationships and node content information within a graph, leading to more efficient and goal-directed clustering results.

At the core of DAEGC is a graph attentional autoencoder, which serves as the foundation for the algorithm's functionality. The graph attentional autoencoder takes the attribute values and graph structure of the attributed graph as input and learns a latent representation by minimizing the reconstruction loss. This process enables the autoencoder to encode the topological structure and node content of the graph into a compact and meaningful representation.

In addition to the graph attentional autoencoder, DAEGC incorporates a self-training clustering module. This module utilizes the learned representation to perform clustering and iteratively refines the clustering results. By generating soft labels from the graph embedding itself, the self-training module guides the clustering process, leading to improved clustering performance.

The key strength of DAEGC lies in its unified framework, where the graph attentional autoencoder and the self-training clustering module work together in a synergistic manner. This joint learning approach allows for the simultaneous optimization of both the graph embedding and the clustering task, ensuring that each component enhances the performance of the other. Overall, DAEGC's architecture and functionality enable it to effectively capture the interplay between graph structure and node content, making it a powerful tool for graph clustering tasks on attributed graphs.

The effectiveness of DAEGC has been demonstrated through experiments on benchmark datasets such as Cora and CiteSeer. Results have shown significant improvements in clustering accuracy and normalized mutual information compared to traditional methods like GAE and VGAE. By integrating both structure and content information, DAEGC outperforms existing algorithms, showcasing its potential for graph clustering tasks.

While DAEGC presents promising results, it is essential to acknowledge its limitations. One notable aspect is the need for careful parameter tuning, especially regarding the dimensionality of the embedding layer. Additionally, the algorithm's performance may vary based on the complexity and size of the input graph data. Further research is required to explore the scalability and generalizability of DAEGC across diverse graph clustering scenarios.

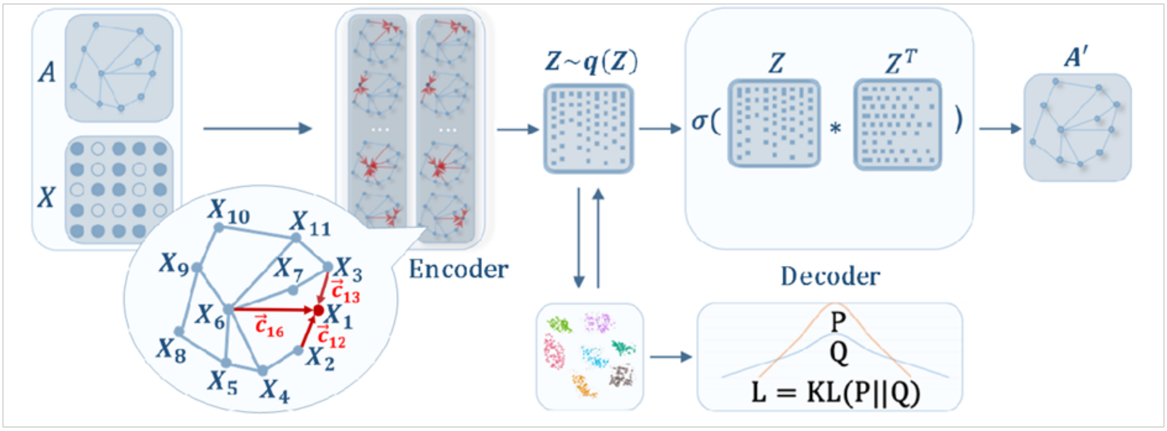


Figure 3.6: Deep Attentional Graph Clustering [54].

3.2.2.4 Deep Dual Graph attention Auto-Encoder for community detection

the integration of high-order modularity information with node attribute data presents a significant challenge, often overlooked by traditional algorithms which typically focus on either structural properties or node attributes, but not both. The Deep Dual Graph attention Auto-Encoder (DDGAE) architecture as shown in figure 3.7) emerges as a pioneering solution to this problem, employing a dual-view approach that effectively captures both types of information. The DDGAE consists of two main modules: the deep dual graph attention auto-encoder, which leverages graph attention mechanisms to learn a unified node representation from both attribute information (**X**) and modularity information (**B**); and a self-training module, which refines community assignments iteratively during the training process, optimizing the detection outcomes.

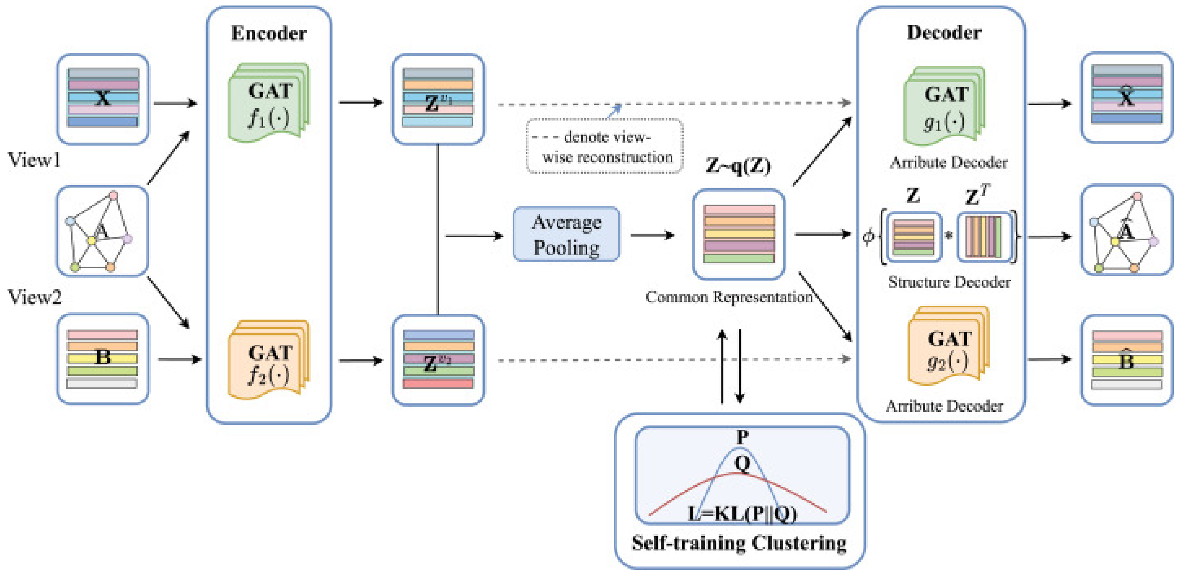


Figure 3.7: Deep Dual Graph attention Auto-Encoder architecture[55].

This sophisticated architecture has been tested on several publicly available datasets, including CORA, CiteSeer, PUBMED, ACM, and DBLP, where it demonstrated notable improvements in accuracy and efficiency over existing state-of-the-art methods. Specifically, DDGAE has shown to enhance community detection performance by leveraging the learned representations to uncover more accurate and coherent community structures within the networks. Despite its advantages, DDGAE's complexity and the computational demands of its dual attention mechanisms present limitations, especially when scaling to very large network datasets.

We present a detailed summary in Table 3.2 of each model architecture used in GAE-based community detection. This table encompasses a summary of various models, highlighting their unique architectural features, underlying input data, and the specific architectures employed in the encoder, decoder, and optimization functions. By providing this detailed comparison, we aim to elucidate the strengths and weaknesses of each approach through an experimental study, which is presented in the next chapter, offering valuable insights into their suitability for different types of networks and community structures.

Table 3.2: Summary of GAE-Based Community Detection.

Year	Abbr.	Full Name	Category	Input	Encoder	Decoder	Loss Function
2018	ARGAE	Adversarially Regularized Graph Autoencoder	Simple	A, X	GCN	Inner product	Reconstruction + Clustering
2018	ARVGA	Variational Adversarially Regularized Graph Autoencoder		A, X	GCN	Inner product	Reconstruction + Clustering + Discriminator
2019	DAEGC	Deep Attentional Embedded Graph Clustering		A, X	GAT	Inner product	Reconstruction + Clustering
2021	EGAE	Embedding Graph Auto-Encoder		A, X	GCN	GCN	Reconstruction + Clustering
2024	DDGAE	Deep Dual Graph Attention Auto-Encoder	Dual	A, X, B	GAT	GAT + Inner product	Reconstruction + Clustering

3.3 Conclusion

In this chapter, we've examined traditional community detection methods alongside prominent algorithms within each. We then briefly explored CNN-based approaches before focusing on autoencoder-based community detection, gaining insight into their mechanics. This understanding sets the stage for our implementation of simple and dual graph autoencoder approaches in the next chapter, where we'll evaluate their performance on datasets and conduct a comparative analysis.

Chapter 4

Experimental Study

In this chapter, we conduct a comparative analysis of graph autoencoder models for community detection, distinguishing between simple and dual encoders across various datasets. We detail the experimental setup and methodologies used, then discuss the comparative results to evaluate the efficacy of each encoder type in identifying community structures.

4.1 Implementation setup

In this section, we will first introduce the datasets utilized in our experiments, followed by a detailed description of the computational environment, configurations used throughout our study, and the evaluation metrics we use to analyze our results.

4.1.1 Datasets

Numerous datasets are commonly used by researchers to study community detection algorithms. For our experiments, we selected three well-known attributed citation network datasets: Cora, CiteSeer, and PubMed. These datasets consist of citation networks where nodes represent documents or authors, and edges denote citation relationships.

The Cora dataset [1] encompasses 2,708 machine learning papers classified into seven distinct classes. The citation network contains 5,429 citations between these papers. Each document is characterized by a binary feature vector that represents the presence or absence of words from a dictionary of 1,433 unique terms.

The CiteSeer dataset [2] consists of 3,312 scientific publications across six classes, featuring a citation network with 4,732 links. Each publication is represented by a binary feature vector that captures the presence of words from a dictionary of 3,703 unique terms.

PubMed [3], a database curated by the United States National Library of Medicine, hosts a wealth of scientific literature, particularly in life sciences and biomedical fields. Its dataset on diabetes comprises 19,717 papers across three classes, with a citation network of 44,338 connections. Each paper is represented by a binary feature vector based on a dictionary of 500 unique words.

Dataset	**# Nodes**	**# Links**	**# Features**
Cora	2,708	5,429	1,433
Citeseer	3,312	4,732	3,703
PubMed	19,717	44,338	500

Table 4.1: Statistical Summary of Citation Network Datasets

4.1.2 Environment

we detail the development environment, including the programming languages and essential libraries that supported our implementation.

[1] `https://paperswithcode.com/dataset/cora`

[2] `https://paperswithcode.com/dataset/citeseer`

[3] `https://paperswithcode.com/dataset/pubmed`

Hardware Setup

The system was equipped with an Intel® Xeon® Silver 4112 Processor featuring a **2.60 GHz** clock speed and **8.25M Cache**. The machine included **32 GB of RAM** and a **512 GB** disk for storage.

Software setup

Python is a versatile, open-source, interpreted language designed to minimize code complexity while expressing programming concepts. It supports both object-oriented and procedural programming styles and boasts an extensive standard library. Python is compatible with most operating systems and is available in the public domain.

Anaconda It is an integrated platform that contains various packages used in data science and machine learning based on Python and R language.

To train the models and conduct advanced analysis, we utilize the following prominent libraries:

TensorFlow [4]is an open-source machine learning library created by Google. It offers a variety of tools and resources for developing, testing, and deploying machine learning models on a large scale.

PyTorch[5] is an open source machine learning (ML) framework based on the Python programming language and the Torch library.

NumPy [6]is a robust Python library designed for scientific computing, offering support for multidimensional arrays, linear algebra, Fourier analysis, and a wide range of other mathematical and scientific functions.

[4]`http://www.tensorflow.org`

[5]`https://pytorch.org`

[6]`www.numpy.org`

Scikit-learn [7]is a widely-used open-source machine learning library for Python, offering robust tools for numerous statistical modeling and machine learning tasks.

NetworkX[8] is a Python library intended for creating, manipulating, and analyzing complex networks and network algorithms.

4.1.3 Evaluation Metrics

In our study, we evaluate the performance of the implemented models using a set of established metrics. Specifically, we utilize the Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), and F1 Score to assess the accuracy and effectiveness of our community detection algorithms

Normalized Mutual Information (NMI)

Normalized Mutual Information (NMI) is a standardized metric for evaluating the performance of community detection algorithms by comparing their output, denoted as C, with a ground truth partition, denoted as C^*. The metric quantifies the shared information between the detected communities and the true communities in a normalized manner. The formula for NMI is given by:

$$NMI(C, C^*) = \frac{-2\sum_{i=1}^{K}\sum_{j=1}^{K^*} n_{ij}\log\left(\frac{n_{ij}}{n_i n_j}\right)}{\sum_{i=1}^{K} n_i \log\left(\frac{n_i}{n}\right) + \sum_{j=1}^{K^*} n_j \log\left(\frac{n_j}{n}\right)} \tag{4.1}$$

K and K^* denote the number of detected and ground truth community counts, respectively, while n is the total number of nodes. n_{ij} signifies nodes shared between the i-th detected community C_i and the j-th ground truth community C_j^*. n_i and n_j represent the node counts in C_i and C_j^*. NMI provides a normalized score between 0 and 1. A score closer to 1 indicates a high similarity between the detected communities and the ground truth[32].

[7]`https://scikit-learn.org/stable`

[8]`https://networkx.org`

Adjusted Rand Index (ARI)

The Adjusted Rand Index (ARI) is a variation of the Rand Index (RI), used to measure the similarity between two clusterings. It considers pairs of points and assesses whether they are placed in the same or different clusters in both clusterings. ARI is calculated using the formula:

$$\mathrm{ARI}(C, C^*) = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_i}{2}\sum_j \binom{n_j}{2}\right]/\binom{n}{2}}{1/2\left[\sum_i \binom{n_i}{2} + \sum_j \binom{n_j}{2}\right] - \left[\sum_i \binom{n_i}{2}\sum_j \binom{n_j}{2}\right]/\binom{n}{2}}$$

where n be the total number of nodes, n_i be the number of nodes in the i-th detected community C_i, and n_j denote the number of nodes in the j-th ground truth community C_j. Furthermore, n_{ij} indicate the number of nodes simultaneously appearing in C_i and C_j [13].

Precision

Precision is a metric commonly used in information retrieval and classification tasks.In the context of community detection, it assesses the accuracy of identified communities by measuring the proportion of correctly clustered nodes within each detected community in comparison to the nodes in the ground truth community.

$$\mathrm{Precision}(C_k, C_k^*) = \frac{|C_k \cap C_k^*|}{|C_k|}, \quad (31) \tag{4.2}$$

where C_k and C_k^* denote the k-th detected community and ground truth community, respectively[31].

Recall

Recall, in the context of community detection, measures the proportion of ground truth nodes within a detected community by counting the common nodes between the detected community (C_k) and the ground truth community (C_k^*) [31]. The formula is given by:

$$Recall(C_k, C_k^*) = \frac{|C_k \cap C_k^*|}{|C_k^*|}$$

F1 score

In community detection tasks,the F1 score strikes a delicate balance between Precision and Recall[31].

$$\text{F1-score}(C_k, C_k^*) = \frac{2 \times \text{Precision}(C_k, C_k^*) \times \text{Recall}(C_k, C_k^*)}{\text{Precision}(C_k, C_k^*) + \text{Recall}(C_k, C_k^*)}$$

4.2 Implementation

This section presents the implementation details of various models evaluated in our study, as outlined in the preceding chapter. These models are categorized based on their encoder architectures into simple and dual encoder types, with a focus on their architecture, key components, and methodologies. Initially, we discuss the common key points for all studied models except for the DDGAE. These models are designed to extract useful representations of nodes in a network, essentially generating compact embeddings through unsupervised learning. Another commonly used technique is K-means clustering, applied to the embeddings to identify community structures. The inputs for this process are the embedding vectors, and the specified number of communities (K).

In the implementation of the Embedding Graph Auto-Encoder (EGAE) model, as detailed in the literature, implemented using the PyTorch library, leverages its dynamic computational graph capabilities to handle the complexities of graph-based deep learning. EGAE features two key graph convolutional networks (GCN) layers: the first transforms raw node features into a 32-dimensional space using ReLU to add non-linearity and extract deep features, while the second compresses these into a 16-dimensional space to enhance computational efficiency and focus on key features for graph reconstruction. The decoder uses an inner product to reconstruct the adjacency matrix from embeddings, effectively predicting node connectivity. Post-training, EGAE applies k-means clustering to these embeddings to identify clusters based on node features and connectivity, employing L1 regularization on GCN weights to prevent overfitting and ensure model generalization.

Building upon the foundational concepts of EGAE, the Adversarially Regularized Graph Autoencoder (ARGA) model is constructed using a dual-layer Graph Convolutional Network (GCN) as the encoder. The first layer of this encoder transforms node features into a 32-dimensional hidden space using a sparse graph convolution that maintains efficiency even with large graphs. It employs the ReLU activation function to introduce non-linearity into the feature space. The second layer further processes these features into a new 32-dimensional space but uses a linear activation to preserve the feature distribution, preparing the embeddings for the reconstruction phase.Its decoder reconstructs the adjacency matrix via an inner product mechanism, enabling effective link prediction between nodes.On the other hand, the Adversarially Regularized Variational Graph Autoencoder (ARVGA)introduces a variational component to the graph embedding process, adding a layer of complexity and robustness. It extends the ARGA architecture by incorporating a third layer that captures the log standard deviations of the latent variables alongside their means. This setup allows ARVGA to model the embeddings as samples drawn from a Gaussian distribution, where the mean and standard deviation are defined by the outputs of the respective GCN layers.Both models incorporate a discriminator that enforces the embeddings to match a predefined distribution, significantly improving their robustness.

Transitioning from adversarial techniques to attention-based methods,the Deep Attentional Embedded Graph Clustering (DAEGC) model introduces a sophisticated architecture specifically designed for attributed graph clustering, which leverages a Graph Attention Network (GAT) as both its encoder and decoder. This model is adept at processing and integrating the dual facets of node attributes (A) and the structural (X) matrix within a graph, promoting an enhanced understanding of community structures through deep learning techniques.

At its core, the DAEGC employs a two-layer GAT encoder that first transforms the input features into a 256-dimensional hidden space and then condenses these down to a 16-dimensional embedding. This sequence not only captures the intricate patterns within the graph through attention-based mechanisms but also ensures the retention of crucial node-specific information by weighting the importance of neighboring nodes differently. The same dimensional structure is mirrored in the decoder, which utilizes the learned embeddings to reconstruct the graph's adjacency matrix. This setup enables the model to predict potential node connections effectively, enhancing the accuracy of the reconstructed graph.This module dynamically adjusts the clustering based on inferred labels from the embeddings, utilizing a loss function designed to minimize the Kullback-Leibler divergence between these labels and their ideal distributions, thereby substantially enhancing the precision of the clustering process.

Transitioning from simple to dual encoder models, the Deep Dual Graph Attention Auto-Encoder (DDGAE) uses a unique dual encoder design with Graph Attention Networks (GAT) to handle node attributes and modularity information, essential for identifying community structures in graphs. Each encoder targets different aspects—attributes and modularity—capturing key similarities and structural details within a 256-dimensional space refined to 16 dimensions. This setup ensures effective emphasis on the importance of neighboring nodes and integrates attribute and structural modularity effectively. DDGAE not only reconstructs the graph's adjacency matrix using an inner product decoder but also the node attributes and modularity matrices, ensuring comprehensive preservation of the graph's topology and detailed community detection. The model enhances its community detection capabilities through a self-training module that iteratively improves clustering by optimizing a combined loss function of clustering and reconstruction losses, with final community assignments further refined using the K-means algorithm.

4.3 Results and Discussion

In this section, we present the findings from our comparative analysis of graph autoencoder models designed for community detection in attributed networks. The models are evaluated based on their encoder architectures—simple and dual—and their performance is quantified using Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), and F-score metrics, as shown in the performance tables for the Cora, CiteSeer, and PubMed datasets (refer to Tables 4.2, 4.3, and 4.4).

Our analysis reveals notable differences in performance between models using dual encoders and those with simple encoders. Dual Encoder models, which process an additional matrix (B) along with the adjacency (A) and feature matrices (X), generally demonstrate enhanced capabilities in capturing complex relational data, particularly evident in the PubMed dataset. The inclusion of matrix B, which likely contains crucial relational or attribute data, appears to enrich the model's data representation, facilitating superior performance across all evaluated metrics in the PubMed dataset.

Moreover, the utilization of the Graph Attention Network (GAT) encoder in both the EGAE and DDGAE models contributes to their improved performance. The attention mechanism within these models allows for a more focused analysis of pertinent nodes or features, which is especially beneficial in large, complex datasets like PubMed.

However, the sophistication of the Dual Encoder models comes with increased computational demands. This is reflected in the execution times illustrated in Figures 4.1a, 4.1b, and 4.1c, where Dual Encoder models generally require more time compared to their Simple Encoder counterparts. This trend is pronounced in the Cora and CiteSeer datasets, where the Dual Encoder model, DDGAE, consistently logs the highest execution times. This increase in computational time is attributed to the additional complexity involved in processing multiple input matrices.

Despite the greater resource consumption, the Dual Encoder model, particularly DDGAE, stands out as the most effective in handling complex datasets, owing to its comprehensive data processing approach. While this model demands more from computational resources, its ability to integrate multiple data types and its detailed processing mechanism yield superior analytical results, making it an appealing option for tasks that demand deep and nuanced data analysis.

Category	**Model**	**NMI**	**ARI**	**F-score**
Simple Encoder	DAEGC	0.5223	0.4844	0.6929
	EGAE	**0.5518**	0.5102	0.5050
	ARGE	0.4892	0.4224	0.6891
	ARVGE	0.3938	0.2850	0.5032
Dual Encoder	DDGAE	0.5423	**0.5243**	**0.7106**

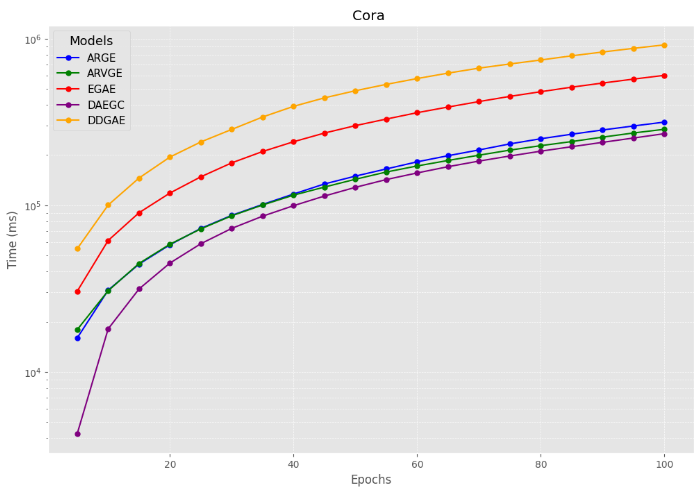
Table 4.2: Models Performance on Cora Dataset

Category	**Model**	**NMI**	**ARI**	**F-score**
Simple Encoder	DAEGC	**0.3665**	0.3423	0.5121
	EGAE	0.2806	0.2720	0.1911
	ARGE	0.3070	0.2936	**0.5888**
	ARVGE	0.3504	**0.3291**	0.5842
Dual Encoder	DDGAE	0.3032	0.2665	0.5038

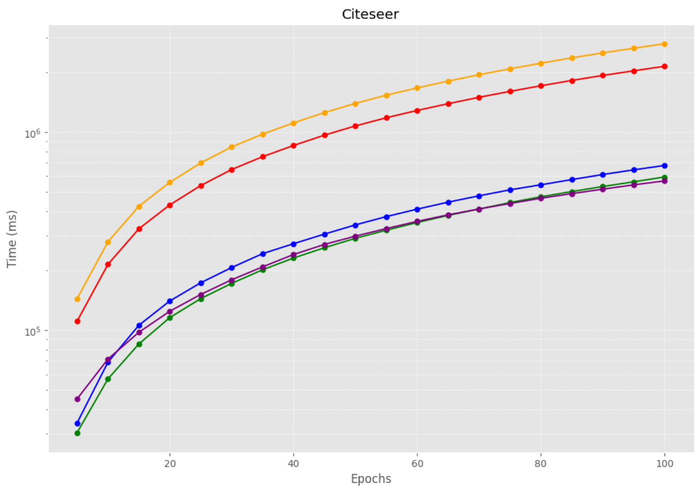
Table 4.3: Models Performance on CiteSeer Dataset

Category	**Model**	**NMI**	**ARI**	**F-score**
Simple Encoder	DAEGC	0.1079	0.0828	0.4639
	EGAE	0.2566	0.2555	0.5833
	ARGE	0.2185	0.2021	0.6173
	ARVGE	0.0443	0.0110	0.3637
Dual Encoder	DDGAE	**0.3004**	**0.3151**	**0.6814**

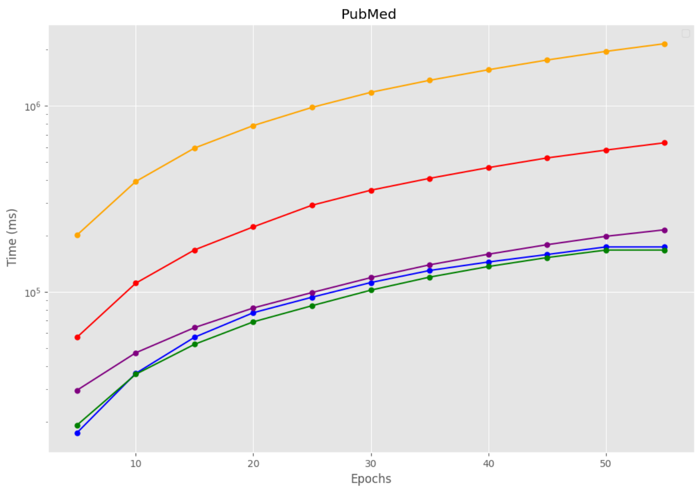
Table 4.4: Models Performance on PubMed Dataset



(a) Cora network.



(b) CiteSeer network.



(c) PubMed network.

Figure 4.1: Models performance comparison across datasets in terms of execution time per epoch.

4.4 Conclusion

In this chapter, we compared graph autoencoder models for community detection using simple and dual encoders on diverse datasets. Through rigorous experimentation and evaluation, we identified the superior performance of dual encoders, especially evident in complex datasets. These findings underscore the importance of encoder architecture in accurately identifying community structures in attributed networks.

Conclusion and Perspectives

Community detection in attributed citation networks is a fascinating and crucial area of research. It's about finding groups of nodes that are more closely connected to each other than to the rest of the network. This task is significant for fields like social network analysis, information retrieval, and bibliometrics. The main question of this thesis is to understand how well different graph autoencoder models can detect these communities.

To explore this, we conducted an in-depth investigation, implementing and testing various graph autoencoder models. We meticulously designed our experiments, utilizing prominent datasets such as Cora, CiteSeer, and PubMed. By evaluating the models with established metrics like normalized mutual information (NMI), adjusted Rand index (ARI), and F1 score, we systematically assessed each model's performance in detecting communities within these citation networks.

Our taxonomy of graph autoencoder models, divided into simple encoder models and dual encoder models, was a crucial element of our research. Our findings indicated that dual encoder models, especially those utilizing Graph Attention Network (GAT) mechanisms, generally provided superior performance compared to the simple encoder models. This was particularly evident with the PubMed dataset, where dual encoder models excelled in capturing complex relational data and community structures. Despite the increased computational resources required by dual encoder models, their enhanced performance underscores their potential for complex network analysis tasks.

While the study faced some challenges, such as varying performance across different datasets and parameter settings, the findings open new research avenues and practical applications. Future research could explore advanced neural network architectures and autoencoders to further improve community detection. Additionally, examining the impact of data preprocessing techniques and model parameters, as well as applying these models to more diverse networks, could enhance their robustness and utility.

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flow Theory, Algorithms, and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] Michael J. Barber and John W. Clark. Detecting network communities by propagating labels under constraints. *Phys. Rev. E*, 80:026129, Aug 2009.

[3] Abdelfateh Bekkair, Slimane Oulad-Naoui, Slimane Bellaouar, Rababe Roumaissa Krimat, and Alla Haddaoui. Cnn-based community detection: A comparison study. In *2023 5th International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, pages 1–6, 2023.

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[6] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. Structural deep clustering network. In *Proceedings of The Web Conference 2020*, page 1400–1410, New York, NY, USA, 2020. Association for Computing Machinery.

[7] J. A. Bondy and U. S. R. Murty. *Graph Theory.* Springer Publishing Company, Incorporated, August 2008.

[8] C. Brouard. *Inférence de réseaux d'interaction protéine-protéine par apprentissage statistique.* PhD thesis, Doctoral Thesis, 2013.

[9] Biao Cai, Yanpeng Wang, Lina Zeng, Yanmei Hu, and Hongjun Li. Edge classification based on convolutional neural networks for community detection in complex network. *Physica A: Statistical Mechanics and its Applications*, 556:124826, 2020.

[10] Mael Canu. *Détection de communautés orientée sommet pour des réseaux mobiles.* PhD thesis, Université Pierre et Marie Curie, 2017.

[11] Liyan Chang and Paula Branco. Graph-based solutions with residuals for intrusion detection: the modified e-graphsage and e-resgat algorithms, 2021.

[12] X. Chen, L. Ma, and X. Yang. Stacked denoise autoencoder based feature extraction and classification for hyperspectral images. *Faculty of Mechanical and Electronic Information, China University of Geosciences*, Accepted 21 June 2015 2016.

[13] Mácio Augusto de Albuquerque, Kleber Napoleão Nunes de Oliveira Barros, Joseilme Fernandes Gouveia, and Rinaldo Luiz Caraciolo Ferreira. Determination and validation of group numbers in a cluster analysis: A case study applied to forestry science. *Acta Scientiarum-technology*, 38:339–344, 2016.

[14] Reinhard Diestel. *Graph Theory.* Springer, 2005.

[15] S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks. *Advances in Physics*, 51(4):1079–1187, 2002.

[16] Jennifer A. Dunne. *Food Webs.* Santa Fe Institute, Santa Fe, USA and Pacific Ecoinformatics and Computational Ecology Lab, Berkeley, USA, 2009.

[17] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[18] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[19] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[20] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proc. of IJCNN, vol. 2*, page 729–734. IEEE, 2005.

[21] Ronald Gould. *Graph Theory.* Dover Publications, Incorporated, illustrated edition edition, 2012.

[22] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. *College of Computer, National University of Defense Technology*, October 26 2017.

[23] A. Gupta, P. Matta, and B. Pant. Graph neural network: Current state of art, challenges, and applications. *Materials Today: Proceedings*, 2021.

[24] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Long Beach, California, USA, 2017. Curran Associates Inc.

[25] William L. Hamilton. Graph representation learning. 2020.

[26] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[27] Dieter Jungnickel. *Graphs, Networks and Algorithms.* Springer Berlin, Heidelberg, 2005.

[28] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[30] G. Liu, Y. Wang, M. A. Orgun, and E. P. Lim. Finding the optimal social trust path for the selection of trustworthy service providers in complex social networks. *IEEE Transactions on Services Computing*, 6(2):152–167, April-June 2013.

[31] Xin Liu, Hui-Min Cheng, and Zhong-Yuan Zhang. Evaluation of community detection methods. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1736–1746, 2020.

[32] Aaron F. McDaid, Derek Greene, and Neil Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. 2013.

[33] C. Merkwirth and T. Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *J. Chem. Inf. Model*, 45(5):1159–1168, 2005.

[34] U. Michelucci. An introduction to autoencoders. `https://doi.org/10.48550/arXiv.2201.03898`, 2022.

[35] Ahmed Ould Mohamed Moctar and Idrissa Sarr. Détection de communautés statiques et dynamiques. *Journal Name (if applicable)*.

[36] B. Mwandau and M. Nyanchama. Investigating keystroke dynamics as a two-factor biometric security. June 7 2018.

[37] D. Müller. *Introduction à la théorie des graphes*. COMMISSION ROMANDE DE MATHEMATIQUE, 2011.

[38] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69:066133, Jun 2004.

[39] Andrew Ng. Sparse autoencoder. Accessed on December 21, 2023.

[40] G. Palla, A.-L. Barabási, and T. Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664–667, 2007.

[41] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding, 2019.

[42] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101:2658–2663, 2004.

[43] F. Radicchi, S. Fortunato, and A. Vespignani. Citation networks. In *Models of Science Dynamics*, page 233–257. 2011.

[44] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007.

[45] Giulio Rossetti and Remy Cazabet. Community discovery in dynamic networks: a survey. Technical report, Knowledge Discovery and Data Mining Lab, ISTI-CNR, Pisa, Italy, February 20 2018.

[46] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, January 2008.

[47] Iqbal H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(Article number: 160), 2021.

[48] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Netw. Learn. Syst.*, 20(1):61–80, 2009.

[49] Robert Sedgewick and Kevin Wayne. *Algorithms.* 4th edition, 2011.

[50] Oleksandr Shchur and Stephan Günnemann. Overlapping community detection with graph neural networks, 2019.

[51] Harmanjit Singh and Richa Sharma. Role of adjacency matrix & adjacency list in graph theory. *International Journal of Computers & Technology*, August 2012.

[52] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[53] Peter Velicković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. 2017.

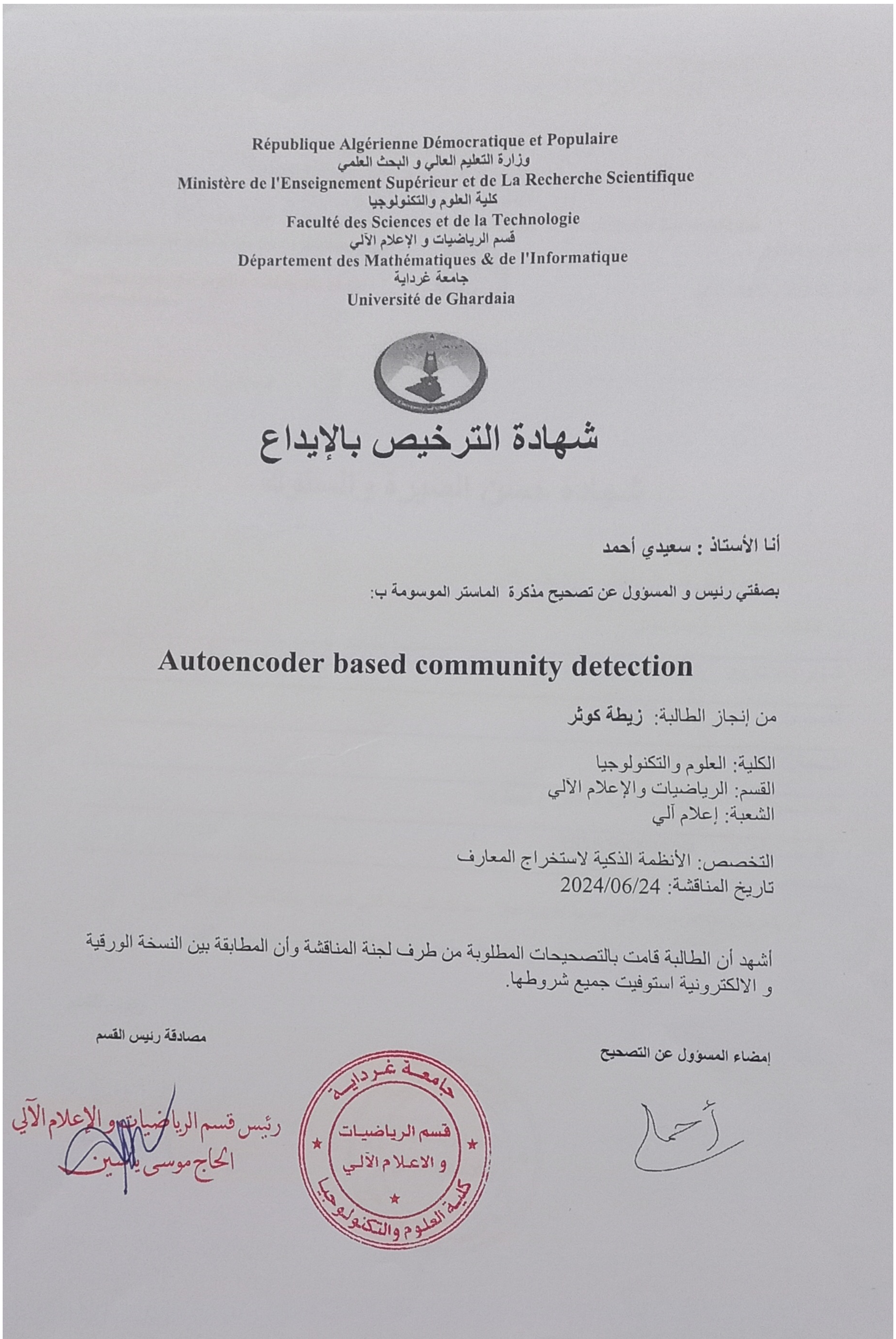
[54] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. Attributed graph clustering: A deep attentional embedding approach, 2019.

[55] Xunlian Wu, Wanying Lu, Yining Quan, Qiguang Miao, and Peng Gang Sun. Deep dual graph attention auto-encoder for community detection. *Expert Systems with Applications*, 238:122182, 2024.

[56] Xin Xin, Chaokun Wang, Xiang Ying, and Boyang Wang. Deep community detection in topologically incomplete networks. *Physica A: Statistical Mechanics and its Applications*, 469:342–352, 2017.

[57] Ilmi Yoon, Rich Williams, Eli Levine, Sanghyuk Yoon, Jennifer Dunne, and Neo Martinez. Webs on the web (wow): 3d visualization of ecological networks on the www for collaborative research and education. San Francisco State University, National Center for Ecological Analysis and Synthesis, Santa Fe Institute, Cornell University, June 2004.

[58] Hongyuan Zhang, Pei Li, Rui Zhang, and Xuelong Li. Embedding graph auto-encoder for graph clustering. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):9352–9362, 2023.



République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique
كلية العلوم والتكنولوجيا
Faculté des Sciences et de la Technologie
قسم الرياضيات و الإعلام الآلي
Département des Mathématiques & de l'Informatique
جامعة غرداية
Université de Ghardaia

شهادة الترخيص بالإيداع

أنا الأستاذ : سعيدي أحمد

بصفتي رئيس و المسؤول عن تصحيح مذكرة الماستر الموسومة ب:

Autoencoder based community detection

من إنجاز الطالبة: **زيطة كوثر**

الكلية: العلوم والتكنولوجيا
القسم: الرياضيات والإعلام الآلي
الشعبة: إعلام آلي

التخصص: الأنظمة الذكية لاستخراج المعارف
تاريخ المناقشة: 2024/06/24

أشهد أن الطالبة قامت بالتصحيحات المطلوبة من طرف لجنة المناقشة وأن المطابقة بين النسخة الورقية و الالكترونية استوفيت جميع شروطها.

مصادقة رئيس القسم

إمضاء المسؤول عن التصحيح

رئيس قسم الرياضيات و الإعلام الآلي
الحاج موسى ياسين

جامعة غرداية
قسم الرياضيات
و الاعلام الآلي
كلية العلوم والتكنولوجيا

[illegible]

[illegible]